Escuela Politécnica Superior

Creación de una App de Gestión para Android con un Framework Actual

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Ramsés Martínez Martínez

Tutor/es:

Jaume Aragonés Ferrero

Universitat d'Alacant
Universidad de Alicante

Junio 2019

AGRADECIMIENTOS

En este apartado me gustaría dedicar unas palabras de profundo agradecimiento a todas aquellas personas que han influido en mi persona durante toda mi vida y que sin todas esas personas posiblemente hoy no estaría escribiendo estas palabras en lo que será el último documento que redactaré en el grado de ingeniería informática.

En primer lugar, a toda mi familia, siendo más precisos a mis padres, Silvia, José y el marido de mi madre, Antonio y a mis abuelos, María y Diego, por su apoyo incansable e incondicional, por el sacrificio que sé que han tenido que hacer para que llegase a la universidad y que pudiese mantenerme en ella, pero sobre todo por ser un ejemplo a seguir en mi vida y con ello haberme hecho crecer como persona, inculcándome valores como el respeto, el compañerismo, la superación personal y el sacrificio por lo que amas.

En segundo lugar, a mi hermano, Absalón, por siempre aportarme el optimismo, la fuerza y las risas que me faltaba para superar momentos y decisiones malas.

En tercer lugar, y no por ello menos importante a mi pareja, Alba, por ser el faro que siempre me ha ayudado a no perderme en este basto mar llamado vida y por siempre aportarme la luz necesaria para llegar a buen puerto. Sin ella, estoy seguro de que no estaría escribiendo estas palabras.

A mis compañeros del grado, que, pese a que nuestros caminos se han ido separando por las clases, por el trabajo o por que han terminado antes, siempre nos hemos apoyado y ayudado en los momentos difíciles, provocando que en ocasiones desearas que el grado no terminara nunca, para poder seguir compartiendo momentos inigualables.

A mi tutor Jaume Aragonés Ferrero, por darme la oportunidad de participar en un proyecto real para una empresa real, con necesidades reales, por su apoyo, consejos, ayuda y paciencia, pero más que cualquier otra cosa, por ser el primer docente que me demostró que la universidad no era únicamente una institución seria y antigua, sino que era algo de lo que tenías que disfrutar, ya que hizo de las clases algo divertido y de lo que obtener una experiencia de vida y profesional, lo cual despertó un gran interés en valorar más lo que estábamos haciendo y en perder el miedo a enfrentarte a cualquier nuevo reto.

A todos los docentes que he tenido, que no sólo me han ofrecido y transmitido conocimientos, sino lo importante y divertida que puede llegar a ser nuestra profesión, pero lo más importante, por transmitirme la pasión que muchos de esos docentes sentían por lo que hacían, que ha provocado que actualmente no sólo me guste lo que hago, sino que sienta verdadera pasión y esté siempre tratando de mejorar y de aprender.

Por último, a la Universidad de Alicante y la Escuela Politécnica Superior, por ser mi casa durante muchos años y de la cual me he llevado experiencias y recuerdos inolvidables e irrepetibles.

ÍNDICE DE CONTENIDOS

AGRADECIMIENTOS **1**

ÍNDICE DE CONTENIDOS **2**

ÍNDICE DE ILUSTRACIONES **4**

INTRODUCCIÓN **6**

OBJETIVOS **7**

ESTUDIO DE MERCADO **8**

IONIC 9
FLUTTER 9
REACT NATIVE 9
NATIVESCRIPT 10
COMPARATIVA ENTRE FRAMEWORKS 10
NiVel de acceso a funcionalidades del sistema *10*
Ecosistema y paquetes/plugins *11*
Popularidad y comunidad *12*
ELECCIÓN DEL FRAMEWORK 13

METODOLOGÍA **14**

SCRUM 14
KANBAN 14
GITFLOW REDUCIDO 15
GITHUB 15
Control del repositorio *15*

ANÁLISIS DE REQUISITOS **17**

INTRODUCCIÓN 17
Propósito *17*
Alcance *17*
FUNCIONALIDADES DEL SISTEMA 18
REQUISITOS FUNCIONALES 18
REQUISITOS NO FUNCIONALES 19
Eficiencia *19*
Seguridad *19*
Usabilidad *19*
Teconológicos *19*
MOCKUPS[27] 20
MODELO DE DATOS (E-R)[27] 27
ARQUITECTURA 29
ENDPOINTS[28] 30

DESARROLLO **31**

API REST CON CONDEIGNITER 32
Seguridad: JWT[31] *32*
Estructura y flujo de datos de la API *33*
ARQUITECTURA Y ESTRUCTURA DE FLUTTER 36
Scoped Model *36*

BLoC Pattern[36] *37*
States Rebuilder[40] *38*
Elección de arquitectura y estructura *39*
LIBRERÍAS Y PLUGINS UTILIZADOS 41
RESULTADOS VS. MOCKUPS 42
Inicio de sesión *42*
Lista de tareas *43*
Añadir parte *45*
Datos del parte *46*
Líneas del parte *46*
Fotos del parte *47*
Firma del parte *49*
PRUEBAS **51**
API REST 51
INTEGRACIÓN E IMPLANTACIÓN **54**
PROBLEMAS ENCONTRADOS **55**
AMPLIACIONES Y MEJORAS **56**
CONCLUSIONES **57**
REFERENCIAS Y BIBLIOGRAFÍA **58**

ÍNDICE DE ILUSTRACIONES

Ilustración 1 - Gráfica de acceso a funcionalidades nativas 10
Ilustración 2 - Gráfico de ecosistema 11
Ilustración 3 - Gráfica de popularidad 12
Ilustración 4 - Imagen de las ramas del proyecto 15
Ilustración 5 - Gráfico con adición y eliminación de código 16
Ilustración 6 - Gráfico de commits por rango de fechas 16
Ilustración 7 - Gráfico de tráfico de commits 16
Ilustración 8 - Mockup login 20
Ilustración 9 - Mockup lista de partes 21
Ilustración 10 - Mockup lista de partes vacía 21
Ilustración 11 - Mockup creación de parte 22
Ilustración 12 - Mockup de lista ordenada 23
Ilustración 13 - Mockup de líneas de parte 24
Ilustración 14 - Mockup detalle de parte 24
Ilustración 15 - Mockup lista de fotos vacía 25
Ilustración 16 - Mockup lista fotos 25
Ilustración 17 - Mockup elección foto 25
Ilustración 18 - Mockup de firma 26
Ilustración 19 - Mockup de firma vacía 26
Ilustración 20 - Modelo Entidad Relación 27
Ilustración 21 - Modalidad Entidad Relación Modificado 28
Ilustración 22 - Arquitectura del proyecto 29
Ilustración 23 - Estructura del código 31
Ilustración 24 - Imagen de ejemplo de JWT 33
Ilustración 25 - Fragmento de código de una ruta 34
Ilustración 26 - Fragmento de método de un controlador 34
Ilustración 27 - Fragmento de código del modelo 35
Ilustración 28 - Estructura del proyecto 39
Ilustración 29 - Fragmento de código de llamada a la API 40
Ilustración 30 - Fragmento de código de inicialización de un widget 41
Ilustración 31 - Inicio de sesión 42
Ilustración 32 - Mockup login 42
Ilustración 33 – Menú emergente 43
Ilustración 34 – Lista de partes vacía 43
Ilustración 35 - Mockup lista vacía 43
Ilustración 36 - Mockup lista partes 43
Ilustración 37 - Mockup lista ordenada 43
Ilustración 38 - Lista partes 43
Ilustración 39 - Parte desplegado 44
Ilustración 40 - Lista de partes ordenados 44
Ilustración 41 - Parte firmado y no firmado 44
Ilustración 42 - Creación de parte 45
Ilustración 43 - Mockup creación de parte 45
Ilustración 44 - Creación de parte 45
Ilustración 45 - Datos del parte 46
Ilustración 46 - Mockup de los datos de un parte 46
Ilustración 47 - Añadir línea 46
Ilustración 48 - Líneas de parte vacías 46

Ilustración 49 - Mockup líneas de parte vacías 46
Ilustración 50 - Mockup líneas de parte 47
Ilustración 51 - Líneas de parte 47
Ilustración 52 – Añadir fotografía 48
Ilustración 53 – Parte sin fotos 48
Ilustración 54 – Escoger fotografía 48
Ilustración 55 – Detalle de la foto 48
Ilustración 56 - Lista de fotos del parte 48
Ilustración 57 – Firmar parte 49
Ilustración 58 – Mockup de parte sin firmar 49
Ilustración 59 – Parte sin firmar 49
Ilustración 60 – Parte firmado 49
Ilustración 61 – Firma del parte 49
Ilustración 62 – Líneas bloqueadas 50
Ilustración 63 – Fotos bloqueadas 50
Ilustración 64 – Foto bloqueada 50
Ilustración 65 – Entornos de Postman 51
Ilustración 66 – Script para automatizar la recogida de tokens y aplicación de estos 51
Ilustración 67 – Consulta exitosa con lista de partes del usuario 52
Ilustración 68 – No se ha proporcionado token 53
Ilustración 69 – Respuesta de token caducado 53
Ilustración 70 – Respuesta de token manipulado 53
Ilustración 71 – Respuesta con la lista de partes vacía 54

INTRODUCCIÓN

Las empresas de hoy en día están viendo incrementada la necesidad de automatizar y agilizar procesos que actualmente se podrían realizar de forma prácticamente autónoma y sin necesidad de tediosos procesos de documentación y verificación, a los cuales solemos llamar *papeleo.* Esta necesidad aparece por la gran cantidad de recursos que se destinan para realizar estas gestiones y que de no tener que hacerlas, se podrían invertir en otros sectores o tareas que proporcionarían un mayor beneficio con respecto a cómo se ha trabajado hasta el momento.

Este proyecto nace de esa misma necesidad, puesto que la empresa en cuestión, la cual ya ha invertido en automatizar y mejorar estas gestiones, ya dispone de una aplicación WEB[1] y un WEB server desarrollados en Codeigniter[2], además de una base de datos en MySQL[3]. Pese a esta inversión, la empresa sigue teniendo una tarea bloqueante y que hace que la automatización del sistema esté comprometida, ya que, tienen que esperar a que los técnicos de la empresa lleguen a la oficina para entregar los partes con la correspondiente información de la resolución de la avería o del servicio proporcionado.

Para cubrir las necesidades de la empresa, se propone desarrollar una aplicación móvil capaz de crear y finalizar partes o incidencias. En dicha aplicación los técnicos tendrán que iniciar sesión con sus credenciales ya existentes en el sistema origen pudiendo de esta manera crear, editar, borrar y terminar los partes que se tengan asignados o se tengan que realizar en esa jornada, pudiendo acreditar el trabajo mediante fotografías y una firma del cliente, donde dicha firma representará la finalización del parte y por tanto dejará de poder ser modificado.

La aplicación consta de dos partes a desarrollar. En el servidor tendremos una API REST[4] desarrollada en Codeigniter. Para guardar los datos tendremos que utilizar la base de datos del sistema original y por último la aplicación móvil desarrollada con Dart[5] mediante el *framework* Flutter[6] utilizando el patrón Scoped Model[7] que es un patrón creado por la comunidad de desarrolladores de Flutter.

Para llevar un control del código y del avance del proyecto se utilizará Git[8] a modo de sistema de control de versiones, utilizando para ello la plataforma GitHub[9]. Se seguirá el flujo de trabajo *gitflow*, pero con una pequeña modificación, y es que la rama de producción no será necesaria, puesto que este proyecto sólo es una pequeña parte de lo que se pretende hacer con la aplicación, por lo que podremos prescindir de ella. Únicamente tendremos una rama de desarrollo de las cuales iremos sacando ramas distintas por cada *issue* o tarea a las cuales se le realizarán *pull requests* para integrarlas con *master* y poder darlas como finalizadas.

OBJETIVOS

Los objetivos generales del proyecto son la creación de la aplicación móvil, a poder ser tanto para Android como para iOS, una API REST para realizar consultas y liberar de lógica de negocio la app y la integración del nuevo sistema con el sistema actual.

Veamos más en detalle los objetivos de cada uno de los objetivos generales del proyecto.

En primer lugar, en este proyecto se propone desarrollar una aplicación móvil, la cual podrán utilizar los técnicos en el mismo momento de realizar el servicio, teniendo la posibilidad de crear en cualquier momento el parte/incidencia rellenando los datos pertinentes y pudiendo de esta manera dejar constancia en todo momento del trabajo que se está realizando y de qué servicios se están proporcionando, además de poder realizar fotografías para acreditar/justificar dichos servicios proporcionados, finalizando el parte con la firma del cliente, la cual, junto a las fotografías se almacenarán en el servidor para posteriormente poder imprimir o generar el PDF con todo el proceso desde que se empezó el parte hasta que se terminó, pudiendo utilizar dicha información como albarán o incluso factura.

En segundo lugar, y no menos importante, se tendrá que implementar una API REST desarrollada en Codeigniter (a petición de la empresa), para que la aplicación móvil se comunique de manera más eficiente y eficaz con la base de datos, utilizando dicha API REST de intermediaria, descentralizando la lógica de negocio a la API y liberando de carga al dispositivo móvil y por tanto también al WEB server.

Por último, tendremos que integrar estas dos partes con el sistema vigente en la empresa, haciendo que todas las partes convivan sin problemas, haciendo que ambos sistemas funcionen entre sí sin que haya ningún tipo de conflicto.

En cuanto a términos de logística de la empresa, quieren poder prescindir de tener que gestionar todos los partes de manera manual, teniendo que insertarlos mediante la aplicación WEB que ya utilizan asiduamente e imprimirlos para posteriormente entregárselos al cliente a modo de albarán o factura.

Por lo tanto, para que este proyecto resulte en entregar valor a la empresa, tendremos que cubrir las necesidades de la empresa, así como las exigencias tecnológicas y logísticas.

ESTUDIO DE MERCADO

Para empezar, hemos realizado un estudio del mercado actual, para decidir qué *framework* usar para el desarrollo de la aplicación, puesto que usar lenguaje nativo, implica que, en un futuro, no podrían adaptar el trabajo realizado a los dispositivos iOS y en menor medida en los Windows Mobile y por tanto un *framework* que nos empaquete la aplicación para que sirva en cualquiera de los sistemas operativos es la mejor opción para que los costes no sean tan elevados, como lo serían si tuviésemos que realizar tres aplicaciones distintas para según qué sistema operativo.

En primer lugar, para realizar la primera parte de la selección hemos buscado los distintos *frameworks* que existen y de todos ellos, hemos escogido los que tengan versiones estables más recientes, además de su repercusión en la comunidad de desarrolladores, dándonos como resultado los siguientes *frameworks* que vamos a pasar a explicar detalladamente.

Para ello, tenemos que entender las tres distintas aproximaciones para desarrollar una aplicación móvil, las cuales vamos a desarrollar a continuación.

- **Utilizar el lenguaje nativo del sistema operativo:** esta aproximación a grandes rasgos es la mejor de todas ellas, puesto que usamos el lenguaje correspondiente a cada sistema operativo, lo cual nos proporciona absolutamente todas las funcionalidades y herramientas del propio sistema, pero como sabemos, tendríamos que aprender distintos lenguajes para poder desarrollar en distintos sistemas operativos.
- **Usar aplicaciones híbridas:** en esta aproximación, empaquetamos o enmascaramos una aplicación web, en una aplicación nativa, mediante el *framework* que escojamos. El más famoso de ellos es Ionic[10] + Cordova[11], los cuales nos proporcionan todas las herramientas y componentes necesarios para enmascarar dicha aplicación web en un navegador oculto, dentro del sistema nativo, por lo tanto, el usuario final realmente piensa que es una aplicación.
- **Usar aplicaciones compiladas:** esta aproximación es la más parecida a desarrollar con un lenguaje nativo, ya que realmente compilamos la aplicación con el código o lenguaje nativo del sistema operativo en el que se instala. Realmente en esta aproximación estamos compilando los componentes de la interfaz de usuario, es decir, los botones, los *containers*, etc, por tanto, esta aproximación funciona significativamente mejor y más rápidamente que la anterior.

Tras ver las distintas aproximaciones, vamos a ver los distintos frameworks que trabajan con la segunda y la tercera, ya que, la primera ha sido descartada, por lo mencionado en esta misma sección.

IONIC

Ionic es un SDK[12] para la creación de aplicaciones híbridas[13], donde se pueden implementar aplicaciones nativas reales mediante el desarrollo de una aplicación web, que posteriormente se empaqueta para que los lenguajes nativos entiendan el contenido, pero realmente estamos realizando una aplicación web enmascarada, básicamente podríamos decir que es un navegador oculto.

Está construido encima de AngularJS[14] y Apache Cordova, lo cual, hace que para un desarrollador web sea un buen *framework* para empezar, ya que realmente estás trabajando con javascript, css y html todo el tiempo.

Es uno de los *frameworks* más usados y por ende con mayor comunidad, ya que nos proporciona una gran cantidad de herramientas que hacen el desarrollo de aplicaciones móviles más fácil y rápido, donde adicionalmente existe una gran cantidad de *plugins* y paquetes que podemos añadir a cualquier desarrollo de una aplicación de este tipo.

FLUTTER

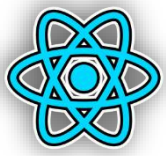
Flutter además de ser un SDK de código abierto, también es un *framework* para Dart, que es un lenguaje de programación desarrollador por Google, al igual que el propio SDK.

Al usar Dart**,** está compilado con librerías de ARM C/C++[15], que trabaja con componentes preconstruidos que podemos ir adaptando a forma de contenedores, para realizar el diseño de la aplicación, que posteriormente se compilará y se lanzará en el sistema nativo, haciendo que la aplicación sea realmente una App nativa y no una web disfrazada.

Dichos componentes se llaman widgets, los cuales son utilizados por Dart y Flutter para construir la interfaz de usuario, pero dichos widgets preconstruidos, sirven para realizar nuestros propios *widgets* personalizados y por tanto más completos.

REACT NATIVE

React Native[16], es una tecnología y un *framework* creado por Facebook, que usa javascript y las librerías de React[17] que te permiten desarrollar una aplicación móvil usando componentes de React para interfaces de usuario.



Además, te permite utilizar paquetes como Redux[18], donde conociendo Javascript y React, hace que sea muy sencillo el empezar a desarrollar la aplicación, ya que realmente estás haciendo lo mismo que cuando realizas un desarrollo web, salvo que, en esta ocasión se compila con el lenguaje del sistema operativo nativo.

NATIVESCRIPT

NativeScript[19] utiliza Javascript para construir las aplicaciones móviles y puede venir desde distintas fuentes como AngularJS, Pure Typescript/Javascript u otros como Vue[20].

Nos permite trabajar con una gran cantidad de *frameworks*, lo cual nos facilita adentrarnos en el desarrollo de aplicaciones móviles, partiendo del conocimiento de alguno de los *frameworks* o lenguajes anteriormente nombrados, además de tener la ventaja de poder elegir los componentes de unos u de otros, según nos convenga.

COMPARATIVA ENTRE FRAMEWORKS

Una vez listados y explicados los distintos *frameworks* candidatos a ser utilizados en el proyecto, podemos realizar de manera totalmente subjetiva y basándonos en nuestra experiencia o en las opiniones de la comunidad de desarrolladores, una elección fundamentada en las necesidades actuales de nuestro proyecto. Para ello, vamos a formular unas métricas que trataremos de explicar y de argumentar de la mejor manera que sea posible, sabiendo de antemano que no es fácil, puesto que nuestra experiencia o la de los demás siempre afecta a la opinión que tenemos sobre ello.

NIVEL DE ACCESO A FUNCIONALIDADES DEL SISTEMA

En la mayoría de las ocasiones, necesitamos acceder a funcionalidades del sistema como puede ser la cámara para realizar fotos, el gps para pintar un mapa o geolocalizar el dispositivo o por ejemplo para generar un pdf y guardarlo en el dispositivo. Por lo tanto, el objetivo de esta métrica es comparar la facilidad con que los *frameworks* explicados acceden a estas funcionalidades.

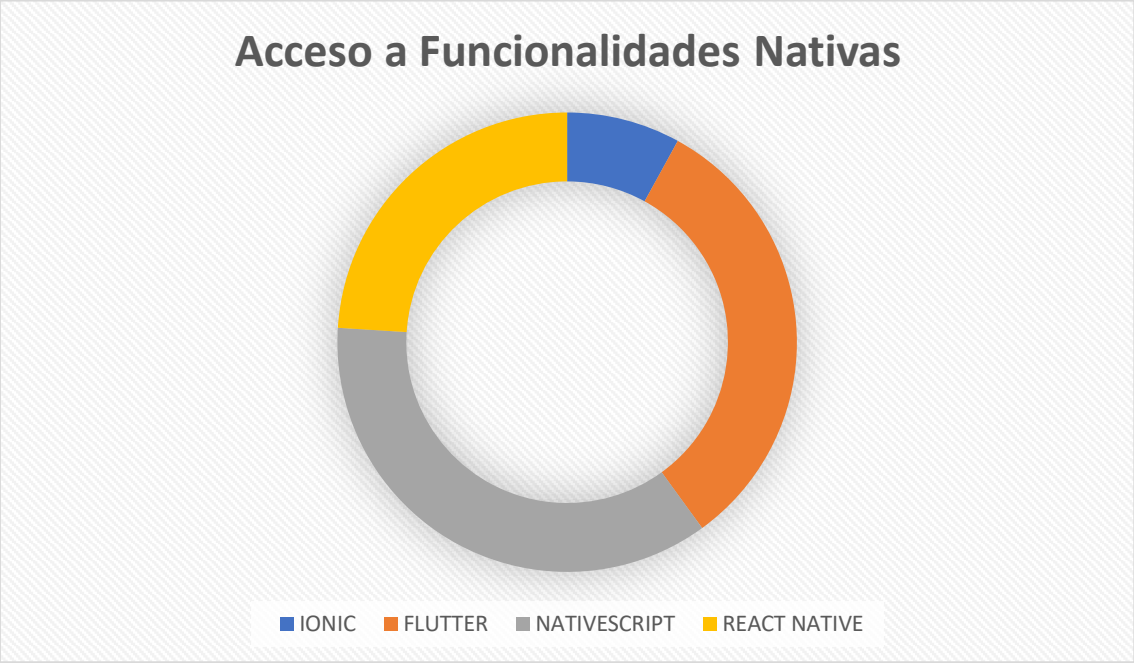


Ilustración 1 - Gráfica de acceso a funcionalidades nativas

Como podemos observar NativeScript tiene la mayor accesibilidad al sistema y la razón que acompaña ese resultado, es el hecho de que contiene una gran cantidad de *plugins* desarrollados por la comunidad, ya que en sus inicios fue una de las mejores opciones para desarrollar aplicaciones móviles, ya que, hay muchos desarrolladores web que utilizan alguno de los lenguajes o *frameworks* que se derivan de la utilización de NativeScript como explicamos en el apartado anterior.

Aun así, Flutter está muy cerca de NativeScript a diferencia de React Native e Ionic, puesto que también existe una gran cantidad de paquetes de terceros, pese a ser el *framework* más reciente, lo cual nos lleva a pensar que muy pronto superará la accesibilidad de NativeScript.

ECOSISTEMA Y PAQUETES/PLUGINS

En esta métrica, vamos a tratar de medir si es fácil encontrar ayuda y si existen librerías o *plugins* que nos extiendan la funcionalidad básica y que podamos incorporar a nuestro proyecto para complacer la demanda de algún requisito concreto.

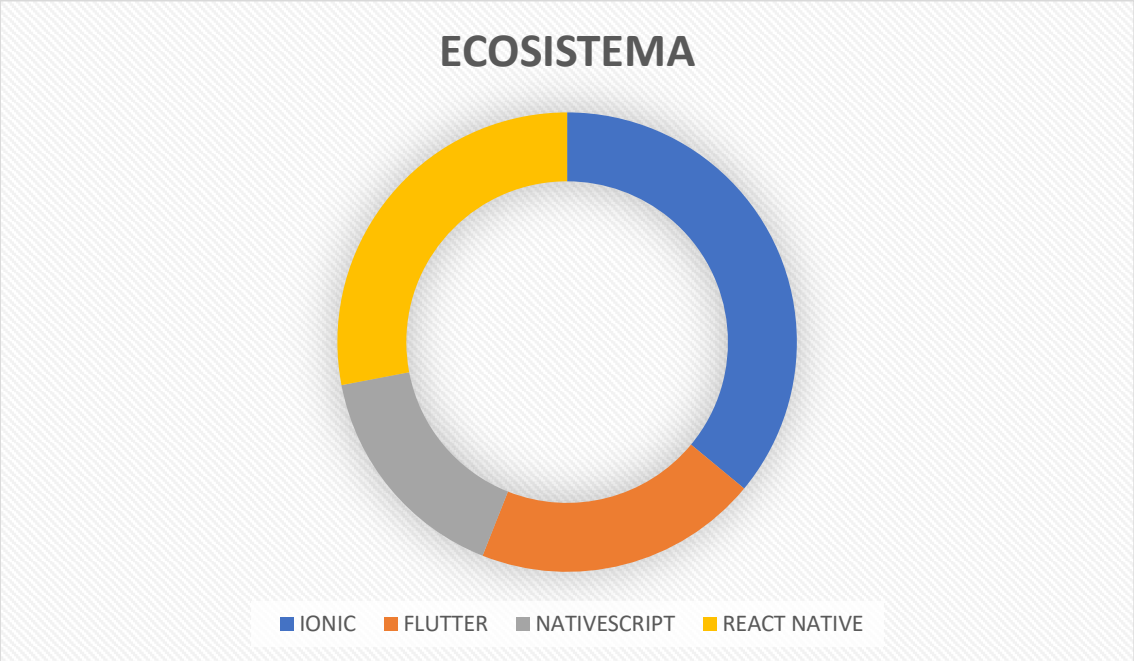


Ilustración 2 - Gráfico de ecosistema

Como era de esperar en esta métrica Ionic sobresale por bastante de entre sus competidores, puesto que es un *framework* muy maduro y con una larga trayectoria con una gran comunidad que la respalda, además de que al usar Javascript es lógico que exista una gran cantidad de paquetes que podemos utilizar o extender del desarrollo web al desarrollo de aplicaciones móviles.

No muy de lejos le sigue React Native, ya que es el *framework* más popular desde no hace mucho para desarrollar aplicaciones móviles mediante Javascript, ya que tiene todas las ventajas de Ionic, salvo que en este caso si se compila con código fuente nativo.

En cuanto Flutter, podemos observar que no tiene un ecosistema tan rico y significativo, pero tenemos que destacar que es un *framework* muy reciente y pese a ello existe una gran cantidad de ayuda y de paquetes y *plugins*, incluso en su propia plataforma que ayuda a que crezca día a día. Todo parece indicar que durante el próximo año va a ser la cabeza visible y la batuta del desarrollo de aplicaciones móviles, según podemos ver en algunos artículos y la gran acogida por la comunidad.

POPULARIDAD Y COMUNIDAD

En cuanto a esta métrica, vamos a tratar de medir la popularidad del *framework* y si existe una comunidad que la apoya y la utiliza, ya que, podemos tener un gran ecosistema que rodee a un *framework* concreto, pero perder el apoyo de la comunidad y la popularidad por algún competidor fuerte con mayores ventajas.

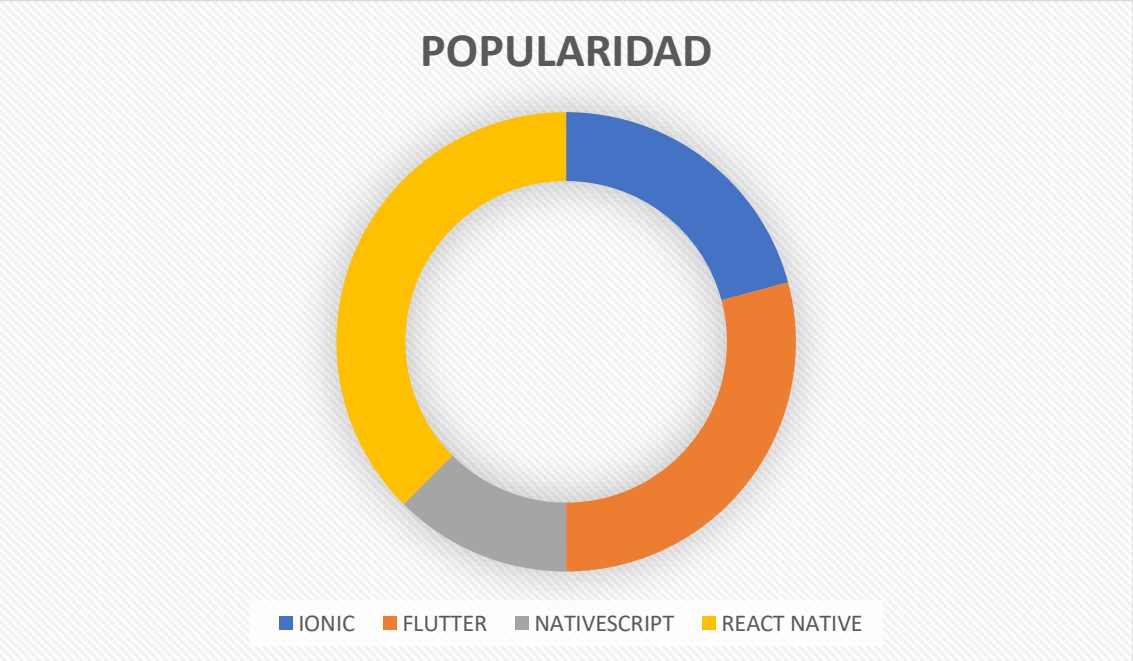


Ilustración 3 - Gráfica de popularidad

En esta métrica, el líder indiscutible es React Native, puesto que utiliza React que es una librería extremadamente popular y el lenguaje de desarrollo web más famoso, es decir Javascript, dándote como resultado aplicaciones nativas reales, lo cual hace posible la inmersión en este tipo de desarrollos, viniendo desde el desarrollo web, obviamente adaptando los conocimientos. Pese a esto, últimamente se está desinflando, ya que cada vez más podemos encontrar más discusiones donde los desarrolladores más veteranos exponen su desagrado por algunas funcionalidades que tras las últimas actualizaciones han dejado de funcionar correctamente y la perdida continuada de comunidad.

Justo detrás tenemos a Flutter, que está creciendo de manera exponencial cada día que pasa, según podemos ver por ejemplo en sus estrellas de GitHub, que hoy en día ya ha superado a Ionic. Además, al estar abanderado por Google, al igual que Facebook con React Native, podemos entender que la popularidad cada vez sea mayor.

ELECCIÓN DEL FRAMEWORK

Como hemos visto en los apartados anteriores, tenemos un gran abanico de posibilidades para escoger y todas esas posibilidades son muy válidas y se convertirían en una aplicación móvil totalmente funcional y conseguida, pero ya que tenemos que escoger una de ellas, tenemos que tratar de escoger la mejor opción para los requisitos que el proyecto nos exige, ya que no estamos construyendo una aplicación desde los cimientos, por qué como ya hemos explicado, existe ya un sistema en la empresa que demanda la aplicación, por lo que tenemos que tratar de adaptarnos al ecosistema existente.

Por lo tanto, está es la elección y las razones por las cuales, creemos fehacientemente que Flutter es la mejor opción para el proyecto que nos ocupa y por qué las demás no lo son.

- **Ionic:** desde el principio fue prácticamente la primera opción, ya que es un *framework* muy maduro y que contiene una gran cantidad de paquetes y *plugins* que nos ayudarían a desarrollar la aplicación, además de que concretamente nuestro conocimiento en el desarrollo web es relativamente amplio, por lo que no resultaría difícil adaptarse a la utilización de este *framework*, pero pese a esto, tenemos que tener en cuenta que Ionic realmente empaqueta una aplicación web y por tanto se trata de un navegador oculto y pese a funcionar relativamente bien, se necesitaría de una conexión a internet estable en todo momento, lo cual, no siempre puede suceder, ya que en este proyecto, se requiere que funcione perfectamente pese a no tener mucha cobertura o incluso sin conexión e Ionic nos dificultaría mucho las cosas en este aspecto.
- **NativeScript:** como hemos podido ver en las distintas métricas, se nota que no es un *framework* popular y tampoco parece que vaya a serlo en un futuro, pese a que Vue está haciendo que el ecosistema esté creciendo paulatinamente. Además, no tiene una gran cantidad de paquetes que nos ayude a extender la funcionalidad.
- **React Native:** esta alternativa, en un principio ni la teníamos planteada, pero tras leer gran cantidad de artículos y de investigar un poco, nos dimos cuenta de que realmente era una muy buena opción para realizar el desarrollo, puesto que su popularidad es indiscutible, además de que trabaja con Javascript y React que al igual que el *framework*, tienen una popularidad y una comunidad extensísima, sin hablar del ecosistema y el nivel y cantidad de paquetes. Perfectamente podríamos habernos decantado por esta opción, puesto que es una de las mejores alternativas.
- **Flutter:** como ya hemos comentado, este será el *framework* con el que desarrollaremos la aplicación, puesto que para empezar, el nivel de acceso a las funcionalidades del sistema es excelente y mucho más rápida y fiable que el resto, puesto que como ya hemos comentado, se trata de un *framework* que se compila con el código y librerías del sistema nativo, además de que tanto su ecosistema como su popularidad crecen de manera exponencial, pero no sólo nos hemos basado en esto para tomar esta decisión, puesto que nos ha motivado el hecho de que pese a ser un *framework* tan reciente, está teniendo una gran repercusión en la comunidad de desarrolladores y sin mencionar que Google está poniendo mucho esfuerzo y dedicación a que funcione y se convierta en la mejor alternativa.

Por lo tanto, tras ver todas las alternativas, investigar sobre ellas y tratar de encontrar sus ventajas y desventajas, Flutter al igual que React Native, son las mejores alternativas hoy en día, ya que tienen un gran apoyo de la comunidad y están en desarrollo continuo y totalmente apoyado por dos grandes compañías que, al estar compitiendo entre ellas, invertirán y harán que ambos *frameworks* tengan una larga y segura trayectoria.

METODOLOGÍA

Para la realización del proyecto haremos uso de las ya tan extendidas y fiables prácticas de las metodologías ágiles[21], que, pese a estar orientadas a equipos relativamente pequeños, las utilizaremos de manera reducida y sin seguir al pie de la letra todos y cada uno de los principios del manifiesto ágil, ya que se centra en las personas, y sería un poco difícil de aplicar en un proyecto donde únicamente interviene un desarrollador.

Para ello, hemos utilizado Scrum[22], Kanban[23], la plataforma Github utilizando gitflow reducido a modo de flujo de trabajo.

SCRUM

Como ya hemos dicho anteriormente, no existe un equipo como tal, por lo que la utilización de Scrum es un poco difusa y no obtendremos todas las ventajas que provienen de dicho proceso, pero, aun así, podemos demostrar que podemos obtener resultados y beneficios de la utilización de este.

Las razones por la cual nos hemos decantado por Scrum son:

- Tenemos que realizar entregas parciales del producto final.
- Tenemos que entregar de manera regular.
- Podemos priorizar según el valor que aportamos a la empresa según sus necesidades y/o urgencias.
- Podemos obtener resultados de manera rápida.
- Podemos detectar errores o discrepancias de la empresa con respecto a la aplicación y realizar cambios sin que afecte de manera excesiva al proceso.
- Puesto que la empresa no tiene muy claro como quiere que sea la aplicación desde un inicio, sabemos que tendremos que ser flexibles y productivos a partes iguales y, por tanto, Scrum cubre esas dos necesidades.

Para aplicarlo, se pactaron iteraciones de entre 2 y 3 semanas, en las cuales se proporciona una entrega parcial del producto final y que es un incremento del resultado final. Las reuniones fueron con el tutor, donde en cada una de esas reuniones le comentábamos los problemas encontrados durante esa iteración y planteábamos posibles soluciones, además de elegir las tareas que entraban en la siguiente iteración.

Además, en cada reunión aprovechábamos para enseñar a modo de demo los progresos realizados y las mejoras introducidas con respecto a la versión anterior y ver si realmente estábamos aportando valor en esa iteración o no.

KANBAN

Escogimos también Kanban por la facilidad de visualizar el flujo de trabajo de manera rápida y sencilla, para poder establecer las tareas que están en progreso y cuales están terminadas o en proceso de prueba. Además, ayudó a planificar y a llevar un mejor control de las tareas que se estaban realizando y las que tenían que realizarse con menor o mayor prioridad según encontráramos tareas bloqueantes o que aportaban más valor al cliente en ese instante.

Nos decantamos por utilizar el propio tablero Kanban de Github, la cual nos parecía más cómodo para tener todo en un mismo sitio.

GITFLOW REDUCIDO

Para el flujo de trabajo hemos utilizado Gitflow, pero utilizando una variante reducida debido a las restricciones del proyecto y de que no somos un equipo y algunas prácticas están definidas para ser utilizadas por un equipo y mejorar el desarrollo paralelo y la colaboración para la integración rápida y sencilla de características nuevas a la aplicación y de esta manera poder aportar valor en cualquier momento, en caso de que la empresa requiera de ello sin que provoque ningún tipo de bloqueo o problema en el proceso.

En este caso, hemos optado por tener únicamente una rama *master*, donde cada rama representa una *issue* y se mezclan mediante un *pull request*, únicamente cuando nos hemos cerciorado de que dicha tarea funcione correctamente.

Para las *releases*, hemos prescindido de tener una rama concreta, ya que Github nos proporciona herramientas para lanzar una *release* en cualquier momento con el código actual de una rama concreta.

GITHUB

Para finalizar, vamos a hablar del control de versiones donde hemos aplicado las prácticas, procesos y metodologías descritas anteriormente.

GitHub es una plataforma de las más conocidas y utilizadas por la comunidad de *developers,* donde podemos realizar el control de versiones de nuestro proyecto y que además abandera y por tanto encaja perfectamente con el flujo de trabajo escogido y las metodologías descritas.

Para poder mezclar con *master*, hemos utilizado los *pull request*, los cuales están vinculados con *issues* concretas y que engloban toda una tarea de Kanban.

CONTROL DEL REPOSITORIO

GitHub nos proporciona una gran cantidad de herramientas que nos ayudan a llevar un control muy completo de todos y cada uno de los repositorios, ofreciéndonos gestionar todas las ramas, *commits*, *issues*, *pull requests* y tareas en el tablero de manera gráfica y muy sencilla. Además, podemos disponer de gráficos muy útiles sobre el rendimiento, la actividad del repositorio, el tráfico, la frecuencia en la que se produce o borra código y demás funcionalidades interesantes. Podemos ver algunos ejemplos en las siguientes imágenes:

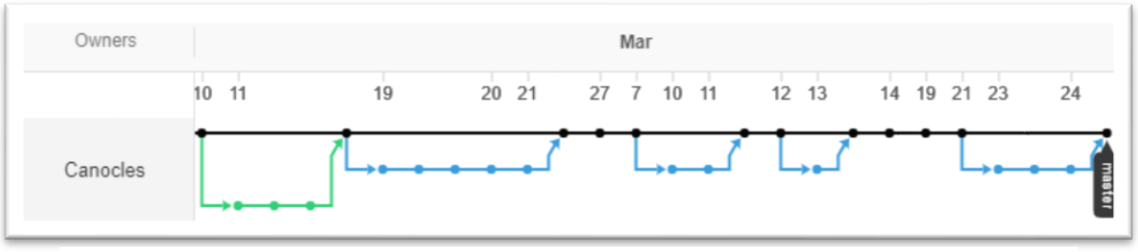


Ilustración 4 - Imagen de las ramas del proyecto

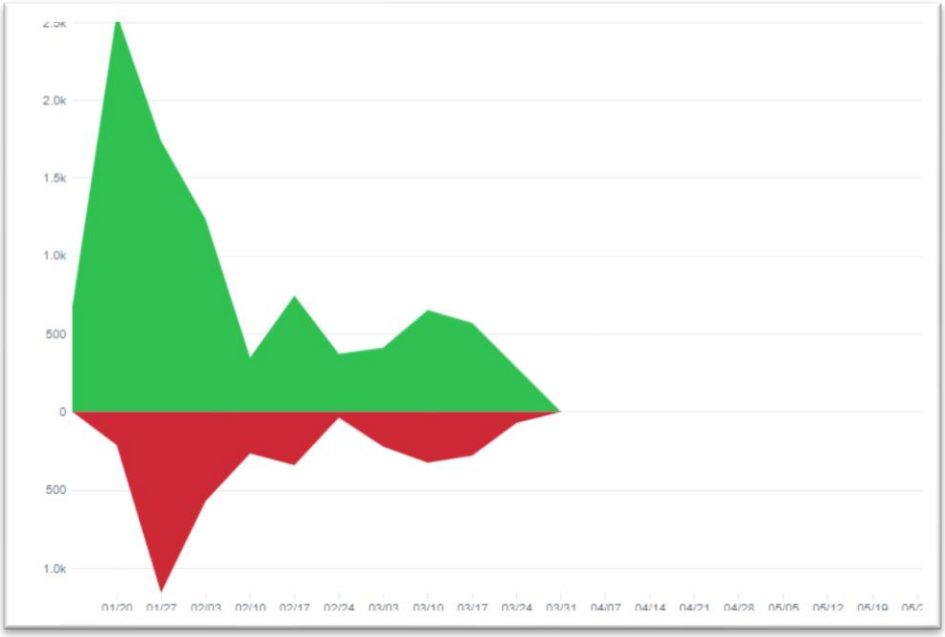


Ilustración 5 - Gráfico con adición y eliminación de código

En cuanto a los *pull request*, están vinculados a una *issue* y dicha *issue* corresponde con una tarjeta en el tablero Kanban, que a su vez corresponde a una tarea escogida para la iteración concreta.

Cuando pasamos a desarrollo una *issue*, creamos de manera manual un *pull request* para que se vincule y de esta manera poder visualizar de manera rápida y sencilla todos los cambios que se están realizando en la rama que se ha creado para esa tarea, ya sean *commits*, cambios de etiquetas, *milestones* o de cualquier otro tipo de información de esa tarea.

Cuando finalizamos la tarea, procedemos a realizar un *merge* con *master* y de esta manera tener ambos cambios en la rama principal y de esta manera disponer siempre de una versión completa, aunque sea parcial del producto final.

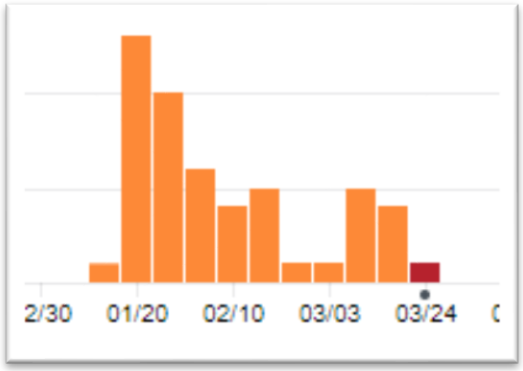


Ilustración 6 - Gráfico de commits por rango de fechas

Ilustración 7 - Gráfico de tráfico de commits

ANÁLISIS DE REQUISITOS

En este apartado, vamos a realizar un análisis de requisitos que posteriormente utilizaremos para hacer un plan de desarrollo que cumpla con todas las expectativas y las necesidades de nuestro cliente.

Dicho análisis ha sido verificado con el cliente, para corroborar que cumple con las expectativas y cubre las necesidades básicas y más urgentes de la empresa.

INTRODUCCIÓN

Softnet[24] nace con el objetivo de romper la barrera entre el usuario y la informática, ofreciendo proyectos de mecanización de procesos de negocio, venta e instalación de cualquier tipo de material informática, asesoramiento en comunicaciones y redes, software de gestión y ERP[25], cursos de formación, mantenimiento de dichos materiales informáticos, entre muchas otras cosas, haciendo de todos estos servicios y ofertas para agilizar las gestiones para la PYME[26].

En cuanto al problema que del cual surge este proyecto, es la necesidad de automatizar y agilizar las acciones de los técnicos, pudiendo hacer uso de una aplicación móvil para realizar el seguimiento de los distintos servicios ofertados, y de esta manera, dejar de lado la habitual burocracia que puede ser útil en un principio, pero que a la larga repercute en la agilidad y el tiempo que se tiene que invertir en pasarlo al sistema informático, con todo lo que ello conlleva. Por lo tanto, poder abrir un parte, realizar la instalación o mantenimiento y cerrar el parte, en el mismo lugar de realización de dicho trabajo es esencial para que la empresa automatice toda la gestión de estas incidencias sin necesidad de trabajar con recibos, justificantes ni tampoco tener que pasar al sistema todo lo que se ha hecho en esa incidencia.

PROPÓSITO

Nuestro propósito es crear una aplicación donde los técnicos puedan de manera rápida y dinámica abrir una incidencia/parte, realizar la instalación, reparación o el diagnóstico de un servicio concreto y dar por terminada dicha incidencia/parte y disponer de todos estos datos en la web privada de la empresa, ya que Softnet ya dispone de una aplicación web y también un *backend* con una base de datos en producción que tendremos que integrar con nuestra aplicación móvil y API.

ALCANCE

En cuanto al alcance de la aplicación, está delimitado por la demanda de la empresa, por lo que para esta versión podremos crear partes, llevarlos a cabo añadiendo líneas y también una fotografía del proceso de manera opcional, y para finalizar el parte, el cliente tendrá que firmarlo para que se complete.

FUNCIONALIDADES DEL SISTEMA

Las funcionalidades principales de nuestra aplicación son las siguientes:

- Inicio de Sesión con las credenciales creadas en la aplicación web.
- Autenticación basada en tokens para garantizar un mínimo de seguridad.
- Integración de la aplicación con sus sistemas vigentes.
- Interfaces sencillas e interactivas que agilizarán el proceso.
- Firma en los partes, para dar por terminada dicha incidencia.
- Soporte para actualizar nuestro software.

REQUISITOS FUNCIONALES

1. **Sistema Operativo**
    - Android.
2. **Conexión con BBDD existente**
    - El sistema tiene que estar integrado con la base de datos actual.
3. **Iniciar Sesión**
    - Podremos iniciar sesión en la aplicación mediante el email y contraseña que dicho técnico tenga asociado desde el sistema actual de la empresa.
4. **Crear Parte**
    - El sistema creará un parte, el cual tendrá que ser cumplimentado con los datos pertinentes. Quedando en estado abierto hasta que sea firmado por el cliente o el técnico lo indique. En caso de que suceda algún error, se mostrará mediante una alerta.
5. **Modificar Parte**
    - El sistema modificará un parte concreto en caso de que exista en la base de datos. En caso de que suceda algún error, se mostrará mediante una alerta.
6. **Eliminar Parte**
    - El sistema eliminará un parte en caso de que exista en la base de datos. En caso de que suceda algún error, se mostrará mediante una alerta.
7. **Listar Partes**
    - El sistema mostrará una lista completa con todos los partes del técnico que ha iniciado, pudiendo filtrarlos por fecha, por estado o por cliente.
8. **Firmar Partes**
    - El sistema permitirá firmar un parte marcándolo como terminado en caso de que exista en la base de datos. En caso de que suceda algún error, se mostrará mediante una alerta.

REQUISITOS NO FUNCIONALES

En este apartado vamos a ver los requisitos no funcionales del producto a desarrollar, concretamente los requerimientos del producto que vamos a desarrollar, los cuales tienen que ser cumplidos para que el producto aporte el valor necesario para el cliente.

EFICIENCIA

- El sistema debe ser capaz de procesar muchas peticiones por segundo, puesto que hay una gran cantidad de técnicos trabajando al mismo tiempo, por lo que debe ser capaz de realizar todas las operaciones sin bloquear el flujo de trabajo.
- Todas las peticiones realizadas al servidor, es decir el API que tenemos que implementar e integrar en sus sistemas actuales, deben responder en menos de 5 segundos.

SEGURIDAD

- Se utilizarán tokens para validar todas las peticiones realizadas al servidor, para asegurarnos de que únicamente puedan realizar peticiones los técnicos con los suficientes permisos.

USABILIDAD

- El sistema tiene que ser muy intuitivo y rápido de utilizar, para que los técnicos puedan finalizar los partes de manera sencilla.
- El sistema proporcionará mensajes de error muy claros, para que el técnico sepa que ha sucedido para poder solventarlo rápidamente.

TECONOLÓGICOS

- El sistema podrá funcionar inicialmente en sistemas Android.

MOCKUPS[27]

Antes de empezar con el desarrollo, como es habitual y más cuando se trata de una empresa real, la cual tiene que utilizar la aplicación, se diseñaron unos bocetos para que dicha empresa validara de manera general las distintas pantallas y el flujo de trabajo con la misma.

Además, vamos a explicar brevemente el flujo de trabajo y las distintas pantallas que podremos ver en la aplicación una vez terminada. A pesar de que representan el flujo de trabajo, se le explicó a la empresa que estos bocetos no son las interfaces finales, por lo que muy probablemente durante el desarrollo cambien ligeramente debido a restricciones tecnológicas o mejoras de diseño.

Ilustración 8 - Mockup login

En primer lugar, al abrir la aplicación veremos el inicio de sesión, donde el técnico tendrá que insertar su correo electrónico y su contraseña, que serán los mismos que tienen en la aplicación web actual.

Puesto que se trata de una aplicación de uso privado para la empresa, se decidió por no implementar un proceso de registro en la aplicación, ya que el administrador del sistema podrá gestionar los técnicos que podrán acceder a la aplicación y con qué credenciales.

Ilustración 10 - Mockup lista de partes vacía

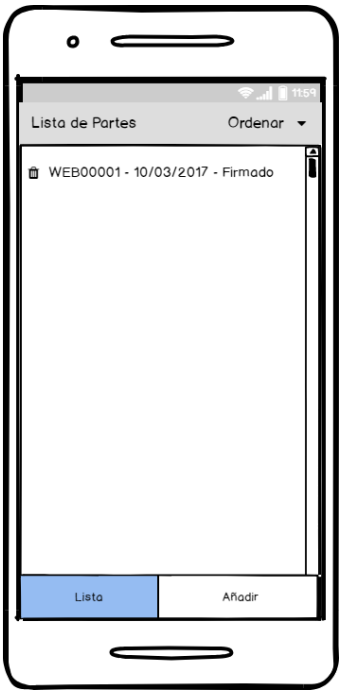


Ilustración 9 - Mockup lista de partes

En segundo lugar, una vez ingresadas las credenciales, si son correctas accederemos a la aplicación, viendo en primer lugar la lista de partes o incidencias que tiene asignadas el técnico que ha iniciado sesión, o los partes que ha dejado a medias y tenga que terminar.

En caso de que no haya ningún parte disponible, aparecerá un mensaje que nos informará de ello, pero en caso contrario, aparecerá la lista de los partes asociados al técnico, además de que para que el usuario sepa que la aplicación está funcionando correctamente, mientras se están realizando las peticiones a la API veremos un *loading* en pantalla para que quede claro que se está procesando la información demandada.

La segunda imagen representa el caso donde el técnico tiene partes asociados, en los cuales existe una serie de acciones que podrá realizar, como puede ser el borrar un parte concreto o visualizar la información del parte pulsado.

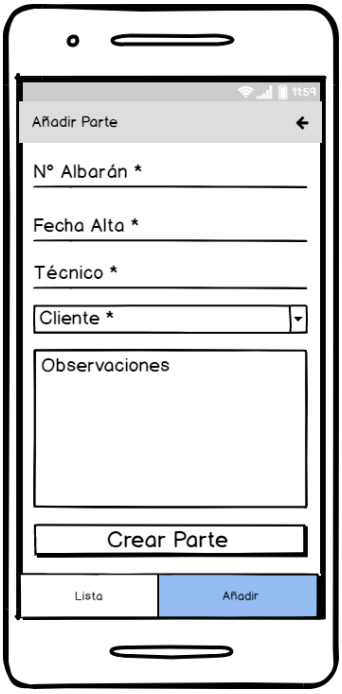


Ilustración 11 - Mockup creación de parte

En tercer lugar, podremos añadir partes nuevos mediante el botón de añadir. Una vez pulsado podremos ver el formulario con toda la información requerida para crear un parte, tanto la obligatoria como la opcional.

Tanto el número de albarán como el técnico vendrán rellenados automáticamente, con el número de albarán que corresponda según la concurrencia en el algoritmo de búsqueda del número mínimo de albarán disponible, donde devolverá el siguiente número de parte a introducir o en caso de que haya algún hueco vacío en la serie de numeración de los albaranes, además del nombre del técnico autenticado respectivamente.

Para crear el parte, tendremos que pulsar sobre el botón de crear parte, y en caso de que algún dato sea obligatorio y no se haya introducido se marcarán en rojo, además, aparecerá un texto debajo de cada campo que no ha pasado la validación con la explicación de lo sucedido.

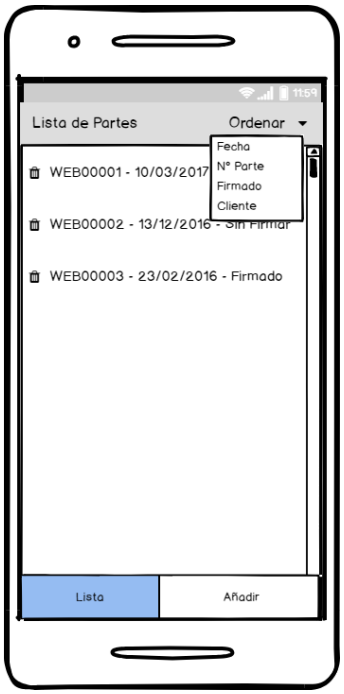


Ilustración 12 - Mockup de lista ordenada

Si se han pasado todas las validaciones del formulario, se procederá a crear el parte mediante la API y nos devolverá a la lista de partes del técnico, donde podremos comprobar que el parte se ha creado con éxito.

También en la lista de partes, podemos ordenar la lista por cuatro propiedades distintas, tanto ascendentemente como descendentemente, donde dichas propiedades son las siguientes:

- Fecha de creación.
- Número de albarán.
- Firmado y no firmado.
- Nombre del cliente.

En el caso de los firmados y no firmados, no habrá un orden ascendente o descendente, sino un filtrado de la lista original devolviendo una copia de lista con los elementos filtrados.

Y en el resto, al pulsar por primera vez ordenaremos de manera ascendente y si pulsamos otra vez, reordenaremos de manera descendente.

Ilustración 14 - Mockup detalle de parte

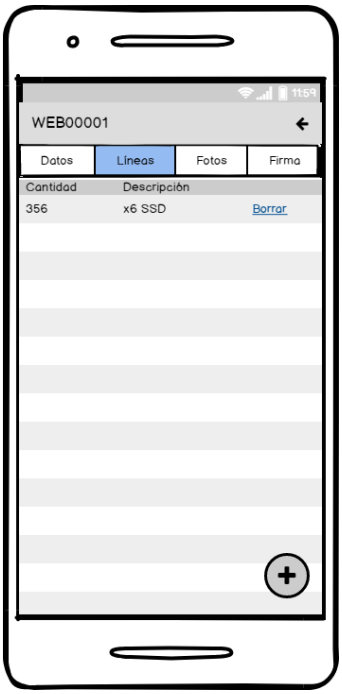


Ilustración 13 - Mockup de líneas de parte

Posteriormente, podemos visualizar el parte que hemos creado, el cual tendrá un menú superior donde podremos navegar por las distintas secciones de un parte, como los datos, líneas, fotos y la firma, además de la habitual flecha para volver a la lista de los partes.

Por lo pronto, tendremos como sección principal los datos principales del parte, donde tendemos la información general del mismo. Dicha sección únicamente es informativa, por lo que no se podrán realizar acciones sobre la misma.

Si queremos visualizar las líneas del parte, podremos ir a la sección de líneas, donde podremos ver, modificar y borrar líneas del parte, siempre y cuando no esté firmado, ya que, en caso de estar firmado, no se podrá modificar ninguna sección del parte, puesto que firmarlo conlleva terminarlo.

Ilustración 15 - Mockup lista de fotos vacía

Ilustración 17 - Mockup elección foto

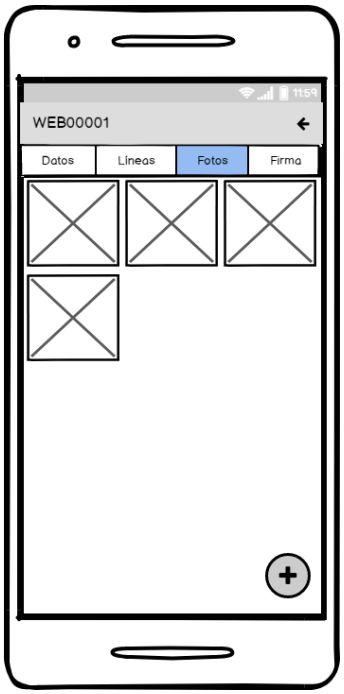


Ilustración 16 - Mockup lista fotos

Podremos también dirigirnos a la sección de fotos, donde podremos añadir y borrar cuantas fotografías se requieran para acreditar y dar fe de los servicios que se estarán proporcionando, pudiendo hacerla en ese mismo instante o escogerla desde la galería.

Lógicamente, también tendremos un mensaje que nos avisará de si se han encontrado o no fotografías o imágenes asociadas al parte seleccionado.

Antes de añadir la foto, podremos visualizar la imagen que se ha escogido o se ha hecho antes de introducirla en el parte, pudiendo descartar dicha imagen en caso de que no nos guste como haya quedado.

Además, podremos pulsar sobre cualquier imagen, para ver la imagen a tamaño real y poder manipularla a nuestro antojo, pudiendo hacer *zoom* o rotándola. En esa misma visualización podremos elegir borrar la foto en caso de que no queramos que esté en el parte, siempre y cuando no se haya firmado.

Ilustración 19 - Mockup de firma vacía

Ilustración 18 - Mockup de firma

Finalmente, tendremos la sección de firma, donde en caso de que existan líneas del parte, podremos pedir al cliente al que se le estén ofreciendo los servicios que firme en el propio dispositivo para que quede constancia de que se ha aceptado los costes y todo el proceso ofrecido. Si no se ha firmado, también nos avisará con un mensaje centrado en la pantalla.

Una vez se registre la firma, el parte se dará por finalizado y, por tanto, ya no se podrá realizar ninguna modificación a ninguna de las secciones del parte, puesto que actuará como un albarán. Si se desea realizar cualquier cambio tras firmarlo, se deberá crear un nuevo parte. Dicha funcionalidad viene requerida por la empresa que utilizará la aplicación.

MODELO DE DATOS (E-R)[27]

En este apartado vamos a mostrar el modelo de datos de la base de datos que nos ha proporcionado el cliente, y del cual tendremos que realizar modificaciones para que la integración con la aplicación móvil sea posible.

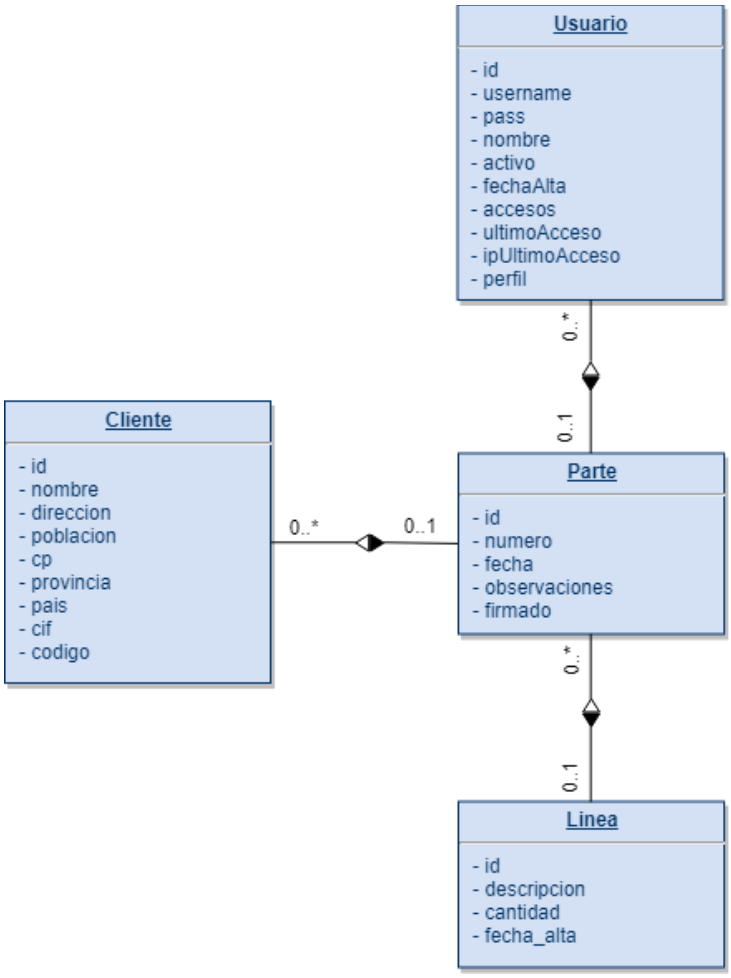


Ilustración 20 - Modelo Entidad Relación

Como podemos observar, tenemos 4 modelos claramente diferenciados:

- **Cliente**: es el modelo que representa los distintos clientes que tiene la empresa
- **Usuario:** es el modelo que representa los usuarios de la aplicación, es decir, los técnicos de la empresa en cuestión.
- **Parte:** es el modelo que representa los partes e incidencias que los técnicos/usuarios tienen que solventar. Además, actúa como albarán para dicha empresa.
- **Línea:** es el modelo que representa las líneas del parte/albarán que se introducirán a la hora de realizar dichos partes.

Tras decidir los requisitos de la aplicación, dicho modelo evolucionó ligeramente para permitir añadir fotos y firmas a los partes, para posteriormente en la web poder generar los PDF con las fotos y la firma del cliente.

En este caso, hemos añadido dos entidades al modelo de datos:

- **Imagen:** es el modelo que representa las distintas imágenes que se han podido realizar a la hora de llevar a cabo el parte o incidencia.
- **Firma:** es el modelo que representa la firma del cliente, que dará por finalizado el parte y, por tanto, se podrá tratar como un albarán en la web.

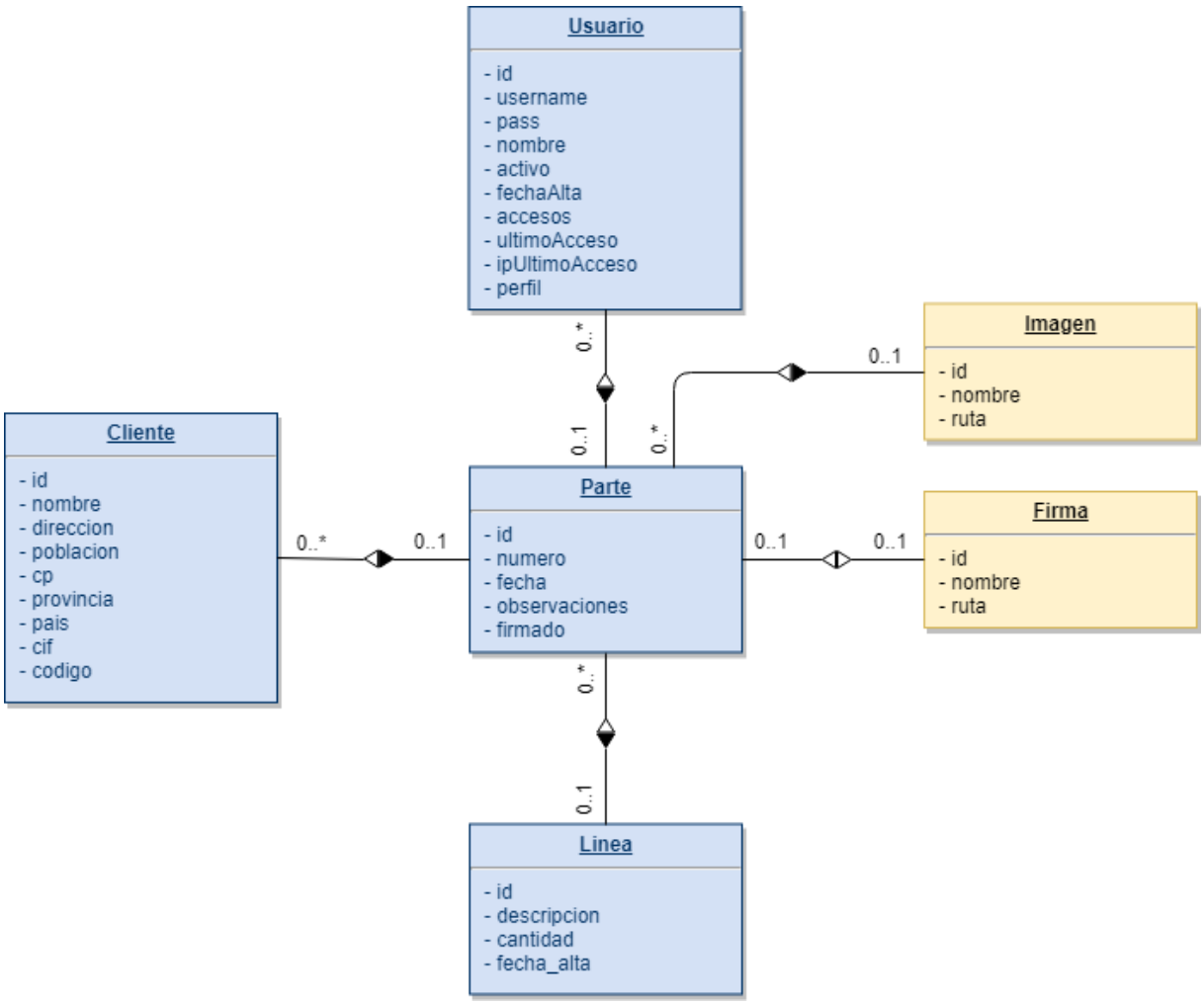


Ilustración 21 - Modalidad Entidad Relación Modificado

ARQUITECTURA

La arquitectura que compone los elementos del proyecto es una de las arquitecturas más habituales y utilizadas del mercado actual, que es la de cliente-servidor.

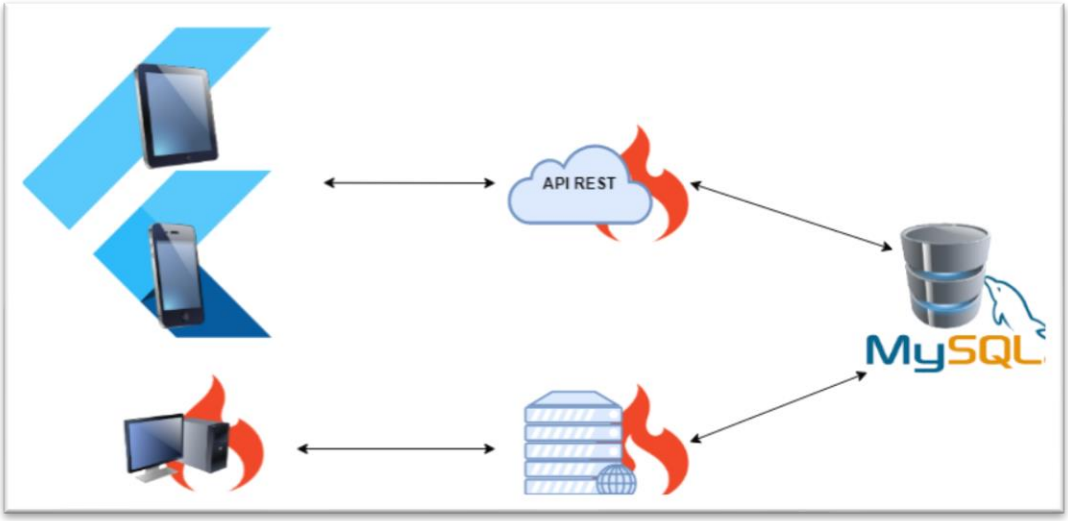


Ilustración 22 - Arquitectura del proyecto

Hasta ahora, la empresa ha trabajado con una arquitectura cliente-servidor de dos capas, donde el servidor accede directamente a los datos y se los devuelve al cliente que realizó la petición, en este caso desde la web ya implementada.

En nuestra propuesta, decidimos añadir una capa intermedia más que se encargara de recoger las peticiones del cliente, procesarlas y enviarlas al servidor de la base de datos para de esta manera repercutir positivamente en cuanto a términos de escalabilidad, seguridad y flexibilidad.

Vamos a ver los distintos componentes más detalladamente:

- **CLIENTES**
  - **APP:** en esta capa, tendremos nuestra aplicación desarrollada en Flutter que se utiliza para crear, empezar y terminar partes, donde todas estas acciones pasan en primer lugar por la API y que posteriormente se guardan en la base de datos, para que el cliente web pueda trabajar con ellos.
  - **WEB:** esta capa ya existía previamente, por lo que en este componente se realizan todas las tareas de gestión de todas las entidades de la base de datos, pudiendo crear, borrar y finalizar partes. En nuestra solución se podrán visualizar los partes realizados desde la aplicación y viceversa.
- **SERVIDORES:**
  - **API REST:** en esta capa, además de servirnos de intermediario entre la base de datos y nuestra aplicación, también tiene parte de la lógica de negocio, para que de esta manera la aplicación no se sobrecargue y quede liberada de realizar conversiones o adaptaciones de la respuesta.
  - **WEB SERVER:** esta capa también existía posteriormente, la cual se encargaba de recibir las distintas peticiones de la web, para posteriormente realizar la petición a la base de datos, finalizando la transacción enviando la respuesta a la web.
- **BBDD:** en esta capa se encuentra el servidor con la base de datos, el cual recibe las peticiones tanto de la API, como del web server, y devuelve los datos pertinentes.

ENDPOINTS[28]

En este apartado vamos a ver los distintos *endpoints* creados en la API para que los dispositivos móviles y/o tabletas puedan consultar y recibir datos. También veremos más detalladamente en apartados posteriores algún ejemplo visual concreto del flujo de datos del sistema implementado.

Todas las rutas comparten una URL base la cual determina el servidor, que en nuestro caso es la siguiente:

- http://softnet/api/v1

Ruta	Método HTTP	Descripción
Inicio de Sesión		
/users/login	GET	Autenticarse
Clientes		
/customers	GET	Obtener lista de clientes
Partes		
/partes	GET	Obtener lista de partes
/partes	POST	Crear un parte
/partes/<id>	GET	Obtener parte
/partes/<id>	DELETE	Borrar parte
/partes/number	GET	Obtener próximo número de parte
Líneas del Parte		
/partes/<id>/lineas	GET	Obtener líneas de un parte
/partes/<id>/lineas	POST	Introducir línea de un parte
/líneas/<id>	DELETE	Eliminar línea de un parte
Imágenes del parte		
/partes/<id>/images	GET	Obtener imágenes de un parte
/images/<id>	DELETE	Eliminar imagen de un parte
/partes/<id>/images	POST	Subir una imagen de un parte
Firma del parte		
/partes/<id>/signatures	POST	Subir la imagen de una firma de un parte

DESARROLLO

En este apartado vamos a centrarnos en hablar de todo lo relacionado con el proceso de desarrollo del proyecto, también hablaremos de algunos elementos de la aplicación móvil que nos han parecido interesantes y que vale la pena explicar más profundamente, además de los resultados finales sometidos a discusión y comparación con los bocetos o mockups diseñados al inicio del proyecto.

Al empezar el proyecto, desconocíamos casi por completo el *framework* que finalmente escogimos para la implementación y desarrollo de la aplicación móvil, además del utilizado para la API, por lo que hemos tenido que pasar por un proceso de aprendizaje de ambas tecnologías, por lo que centremos la atención en este punto.

En primer lugar, al realizar el estudio del mercado ya nos introducimos bastante a cómo funcionaba Flutter, además de sus ventajas y desventajas, lo cual ayuda bastante a saber por dónde empezar y qué aspectos del *framework* nos vendrán mejor para el proyecto.

Para empezar a aprender el *framework* realizamos unos cuantos ejercicios con ejemplos que la propia documentación[29] de Flutter nos ofrece, para crear nuestras primeras aplicaciones de prueba y poder de esta manera aprender de manera práctica, que en nuestra opinión es la única forma de afianzar con seguridad y rapidez cualquier lenguaje de programación como es Dart o cualquier *framework*.

Una vez realizados nuestras primeras aplicaciones de prueba, decidimos decantarnos por comprar algún curso que nos ofreciese mayor profundidad de conocimientos. En nuestro caso, compramos varios cursos sobre Dart y Flutter en Udemy[30], donde pudimos realizar hasta cuatro aplicaciones distintas antes de poder empezar a trabajar con el proyecto, ya que, queríamos ofrecer a la empresa un producto lo más profesional y completo posible, siempre ajustándonos a sus necesidades y exigencias.

El aspecto que más nos sorprendió de Flutter es su versatilidad y su curva de aprendizaje prácticamente nula al inicio, pero que cuando buscas funcionalidades más completas y complicadas, puede convertirse en un reto, por la poca madurez del *framework*. Pero pese a ello, se pueden hacer aplicaciones muy conseguidas sin tener que profundizar mucho en los conocimientos con más coste en cuanto al aprendizaje.

Además, si es la primera vez que trabajas con Dart y Flutter, te extraña en demasía la forma en que se estructuran las líneas de código, puesto que al menos en nuestro caso, nunca habíamos visto esta estructura, como podemos ver en la imagen de la derecha.

```
Widget _buildCliente() {
  return Padding(
    padding: EdgeInsets.only(left: 15.0, right: 15.0),
    child: TextFormField(
      initialValue: parte.customer.name,
      decoration: InputDecoration(labelText: 'Cliente'),
      enabled: false,
    ), // TextFormField
  ); // Padding
}

Widget _buildTecnico(MainModel model) {
  return Padding(
    padding: EdgeInsets.only(left: 15.0, right: 15.0),
    child: TextFormField(
      initialValue: model.user.name,
      decoration: InputDecoration(labelText: 'Técnico'),
      enabled: false,
    ), // TextFormField
  ); // Padding
}
```

Ilustración 23 - Estructura del código

Si observamos detenidamente, podemos ver como los componentes, o en caso de Flutter, los widgets, están formados por el nombre del widget y entre paréntesis las distintas propiedades de dicho widget, pudiendo encapsular widgets dentro de otros widgets para poder personalizar a nuestro antojo cualquiera de los widgets que vienen por defecto, además de los que podemos crear nosotros mismos como podemos ver en la imagen.

API REST CON CONDEIGNITER

Como ya hemos avanzado en apartados anteriores, hemos implementado una API REST desarrollada en Codeigniter debido a requerimientos tecnológicos por parte de la empresa, ya que todo su sistema actual está desarrollado en dicho *framework*, y, por tanto, será mucho más fácil de mantener por parte del técnico de la empresa.

Por tanto, al igual que ha sucedido con la aplicación móvil, hemos tenido que pasar por un periodo de aprendizaje para poder desarrollar el API y también integrarlo con su sistema actual.

Inicialmente, decidimos modificar el servidor web para integrar en ese mismo servidor las nuevas funcionalidades, pero tras investigar las tecnologías y versiones que se habían utilizado en el sistema decidimos mantenerlo como dos sistemas diferenciados en distintos servidores, puesto que la versión en la cual se había desarrollado el sistema era suficiente antigua como para que la integración de la API en ese mismo sistema habría supuesto un esfuerzo mayor, el cual no representa beneficio alguno.

Finalmente hemos decidido por separar el API del servidor web, como ya hemos podido ver en el apartado de análisis de la arquitectura de todo el proyecto, para que de esta manera podamos tener dos capas diferenciadas y que puedan convivir entre ellas sin necesidad de realizar grandes esfuerzos para adaptar ambos sistemas a un punto intermedio donde ambos puedan coexistir.

Para el desarrollo de la API, hemos utilizado una librería de terceros, la cual es una de las más utilizadas en la versión 3.0, ya que por defecto trae la estructura y los *middlewares* necesarios para poder implementar un API Rest de manera sencilla de entender y de desarrollar, dada la poca experiencia con el *framework*.

Cabe destacar, que, en el momento del desarrollo de la API, el *framework* no proveía de funcionalidades propias para poder implementar una API y tenías que depender de librerías de terceros, pero actualmente ya dispone de un estándar que también facilita el desarrollo de una API.

SEGURIDAD: JWT[31]

En este apartado vamos a ver por encima las medidas de seguridad para los *endpoints*, para asegurar que únicamente los usuarios autorizados puedan realizar peticiones y que no se pueda acceder de manera externa.

Para ello, hemos utilizado uno de los estándares más utilizados, puesto que es un estándar abierto basado en JSON[32] para la creación de tokens[33] de acceso que permiten la propagación de identidad y privilegios.

Básicamente, se trata de la creación de una identidad temporal para que los usuarios puedan acceder a los recursos que requieren de token para poder acceder, eligiendo la caducidad de dicha identidad para de esta manera controlar que no se ha suplantado dicha identidad y provoque problemas de seguridad que pueden ser graves en el caso de una aplicación de gestión que maneja datos sensibles de empresas y de clientes.

JWT se compone de tres partes diferenciadas:

- **Header:** es el encabezado del token que se compone a su vez por el algoritmo de firmado y el tipo de token.
- **Payload:** es el contenido del token y contiene la información de los privilegios del token.

- **Signature:** es la firma del token, que se calcula mediante la codificación de la cabecera y el contenido en Base64[34], conectándose ambas partes por un punto.

Una vez definidas las tres partes, se codifican de nuevo en Base64 cada una de las partes y las separamos por puntos.

En la página web de JWT podemos ver un ejemplo gráfico sobre las distintas partes de un token y también cómo se combinan para formar un token.

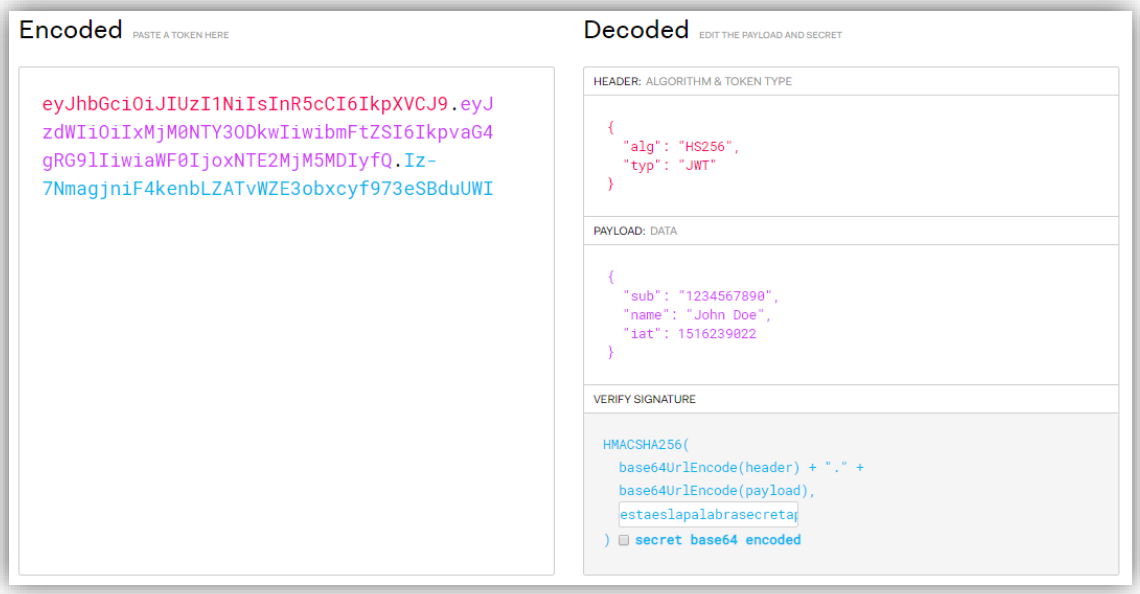


Ilustración 24 - Imagen de ejemplo de JWT

ESTRUCTURA Y FLUJO DE DATOS DE LA API

Vamos a profundizar en cómo hemos desarrollado los *endpoints* de la API, mediante un ejemplo visual de uno de ellos paso a paso.

En primer lugar, la estructura de la API dispone de un archivo de configuración global para toda la aplicación, pero también disponemos de un archivo de configuración para las rutas de la aplicación, las cuales serán las que se consultan a través de la aplicación móvil, donde dichas rutas están apuntando a un controlador que se encarga de recibir y procesar los datos de la petición y llamar a un método dentro del modelo que se encargará de realizar las peticiones a la base de datos, y devolver los datos de forma que podamos procesarlos para devolverlos a la aplicación, ya sean errores o datos concretos.

Para que sea más sencillo de explicar, vamos a partir de un ejemplo concreto para ver cómo funciona el flujo de datos de y como se va transformando la información desde que viene desde la aplicación móvil hasta que vuelve a la misma.

El primer paso es crear una ruta para que los clientes, en este caso la aplicación móvil pueda llegar hasta la API para que posteriormente se realicen acciones.

```php
/**
 * Imágenes del Parte
 */
$route['api/v1/partes/(:num)/images']['get'] = 'parte_api/getImagenesParte/$1';
```

Ilustración 25 - Fragmento de código de una ruta

Como podemos observar, la ruta está compuesta por dos partes, las cuales están diferenciadas por las llaves que las encapsulan. La primera de ellas es la ruta que habilita el acceso a un método concreto del controlador, el cual está en la parte derecha de la asignación, donde podemos intuir que se dirige al controlador de partes para ejecutar el método llamado *getImagenesParte* y le pasa un parámetro, que en este caso concreto, es el primer y único argumento que podemos ver en la ruta, que según las buenas prácticas de una API Restfull[35], debe ser el identificador del parte, al cual vamos a devolver todas las imágenes de dicho parte. En cuanto a la segunda parte de la ruta, tenemos el método HTTP que vamos a realizar, que en este caso es un GET.

Cuando se realiza una petición a dicha ruta, nos dirige directamente al controlador y llama al método nombrado.

```php
public function getImagenesParte($id) {
    header("Access-Control-Allow-Origin: *");

    $this->_apiConfig([
        'methods' => ['GET'],
        'requireAuthorization' => true,
    ]);

    $parte_imagenes = $this->parte_model->getImagenesParte($id);

    if ($parte_imagenes == 'INVALID_PARTE') {
        $this->api_return(
            [
                'error' => $parte_imagenes
            ],
        404);
        return;
    } else if ($parte_imagenes == 'EMPTY_IMAGES_FROM_PARTE') {
        $this->api_return(
            [
                'message' => $parte_imagenes,
                'imagenes' => []
            ],
        200);
        return;
    } else {
        $this->api_return(
            [
                'parte' => $id,
                'imagenes' => $parte_imagenes
            ],
        200);
        return;
    }
}
```

Ilustración 26 - Fragmento de método de un controlador

Como podemos observar, el primer paso y el más importante es comprobar que se está enviando un token y que dicho token sea correcto y que además no haya expirado, en caso de que sea válido y no haya expirado seguiremos el flujo normal del método, pero en caso contrario devolverá un mensaje de error indicando qué ha sucedido con el token.

Posteriormente realiza una llamada al modelo para que el modelo haga la llamada a la base de datos y trate de encontrar las imágenes de dicho parte.

```php
public function getImagenesParte($id)
{
    if (is_null($id)) {
        return 'INVALID_PARTE';
    }

    $query = $this->db->select('*')->from('softnet_imagenes')->where('parte', $id)->get();
    if ($query->num_rows() > 0) {
        return $query->result_array();
    }

    return 'EMPTY_IMAGES_FROM_PARTE';
}
```

Ilustración 27 - Fragmento de código del modelo

En esta última instancia, comprobaremos que el id no sea nulo, para no disparar un error de base de datos, el cual no podemos controlar tan fácilmente.

Una vez comprobado que no sea nulo, realizamos la petición a la base de datos y en caso de encontrar alguna imagen, devolveremos un array con todas las imágenes encontradas y en caso contrario devolvemos una cadena representativa de no haber encontrado ninguna imagen de ese parte.

Finalmente, en el controlador de nuevo, en caso de que no se hayan devuelto cadenas representativas de errores, y exista un array con datos, devolveremos dicho array al cliente, el cual tendrá que procesar la respuesta y realizar las acciones pertinentes.

Por lo tanto y para concluir este apartado, podemos ver como en el proceso únicamente estamos interactuando entre el cliente, la API y la base de datos sin ningún momento tener necesidad de interactuar con el servidor web y la aplicación web que ya existe en su sistema, haciendo que no afecte al rendimiento y liberando de una gran carga al servidor web y a la aplicación móvil.

ARQUITECTURA Y ESTRUCTURA DE FLUTTER

Al igual que ocurre con tantos otros *frameworks* o lenguajes de programación, antes de ponernos a escribir código, tenemos que tener claro qué arquitectura y cómo vamos a estructurar el código, por lo que vamos a discutir sobre las distintas arquitecturas presentes en el mercado y las razones por las cuales hemos escogido una de ellas.

Para empezar, debemos aclarar que Flutter es declarativo, puesto que reconstruye las interfaces desde cero, es decir, las repinta cada vez que el estado de dicha interfaz cambia para de esta manera cambiar de manera dinámicamente partes de ese componente.

El problema más común que nos encontramos con Flutter una vez empezamos a utilizarlo para más de un par de pantallas y teniendo que relacionarlas entre sí es el cómo, dónde y cuándo tenemos que realizar una llamada para cambiar el estado de alguna variable o componente concreto. Para responder a esas cuestiones Flutter nos ofrece una solución *Vanilla* que nos puede servir para cambiar el estado de pequeñas porciones de código y sin tener que realizar cambios a interfaces embebidas o anteriores, pero normalmente nos encontraremos con situaciones que requieran de un cambio de estado mucho más complejo, como puede ser desde una pantalla cualquier cambiar los estilos de color de toda la aplicación, lo cual sería muy tedioso y enrevesado realizarlo con el patrón *Vanilla* que propone Flutter.

Para ello, la comunidad ha creado distintos patrones que nos ayudarán al manejo, almacenamiento y control del estado en nuestra aplicación. Concretamente vamos a ver en más detalle tres de ellos, los cuales únicamente uno de ellos no ha sido creado por la comunidad, pero tampoco es el uso estándar, sino que hacemos uso de funcionalidades del sistema avanzadas para realizar el control del estado.

SCOPED MODEL

Según sus propios creadores, esta librería nos ayuda a controlar los datos y estados de nuestra aplicación de manera ordenada, consiguiendo así tenerlos separados de la lógica de las UI. Facilita la transmisión del modelo de datos entre un Widget padre hacia sus descendientes. Además, también reconstruye todos los hijos de ese Widget padre cuando cualquier el modelo cambia.

Para entrar más en detalle, tenemos tres clases principales que utilizaremos para definir el modelo, el widget que actuará como padre y los widgets que actuarán como hijos y que podrán cambiar el estado en cualquier momento, provocado que todo el árbol se reconstruya.

En primer lugar, tenemos el **Model**, que representará el modelo de datos que queremos que contenga el estado y los datos concretos que queremos poder controlar entre distintos widgets.

En segundo lugar, tenemos el **ScopedModel**, que se trata de un widget que se encargará de traspasar el modelo hacia abajo en cuanto a jerarquía. Podemos encapsular el modelo dentro del ScopedModel provocando que el modelo esté disponible para todos los descendientes del Widget.

Por último, tenemos el **ScopedModelDescendant**, que se utiliza para encontrar el ScopedModel concreto en el árbol de componentes y que se encargará de notificar hacia arriba a todos los descendientes del ScopedModel encontrado.

Podemos entonces definir las siguientes ventajas:

- Lógica, interfaces y control de estado separados.
- Fácil de aprender.

- Fácil de aplicar en aplicaciones pequeñas o medianas.

En cuanto a las desventajas tenemos las siguientes:

- Es una liberaría de terceros, lo cual significa que podría dejar de ser mantenida en cualquier momento.
- Si se utiliza en aplicaciones muy grandes donde el modelo crece exponencialmente, conforme más complejo se vuelve el modelo más difícil es tener claro donde realizar la llamada para que se reconstruya el árbol de componentes.

BLOC PATTERN[36]

BLoC (**B**usiness **Lo**gic **C**omponents) es el patrón más recomendado y usado por la comunidad, ya que nos ofrece flexibilidad tanto para aplicaciones medianas como grandes.

Básicamente, es un sistema de gestión del estado, ayudando a la gestión y acceso de este. Existen patrones muy parecidos en otras tecnologías, como podría ser Redux en React. Consiste en tener un almacén central de estados, al cual podemos acceder desde cualquier punto de la aplicación para consultar, modificar o crear estados, donde dicho almacén es consultado por las interfaces de la aplicación.

Podemos leer un artículo de Sagar Suri[37], donde explica mucho más a fondo como se utiliza y ejemplos prácticos.

Podemos entonces definir las siguientes ventajas:

- No es una liberaría de terceros, lo cual, podremos seguir utilizando el patrón, ya que no se tiene que mantener.
- Lógica, interfaces y control de estado separados.
- Es reactivo, por lo que no tenemos que realizar llamadas para dar a conocer que el estado ha cambiado.

En cuanto a las desventajas tenemos las siguientes:

- Conocimientos de streams[38] y rxdart[39] necesaria para la comprensión del modelo.
- Curva de aprendizaje alta si no se ha trabajado con los elementos anteriores.
- Crea mucha lógica y clases para poder realizar un control exhaustivo de los distintos estados.
- Complicado de utilizar en aplicaciones pequeñas.

STATES REBUILDER[40]

Este patrón y plugin aún no es muy conocido y puede ser difícil de encontrar en un inicio con patrones tan famosos como los antes nombrados, pero posiblemente sea la mejor opción para empezar a entender los ciclos de vida de Flutter y manejar de manera eficiente y sencilla el estado de la aplicación en todo momento.

Decimos que es la mejor opción, porque nos proporciona la flexibilidad y potencia de BLoC, pero sin la complejidad de este, ofreciendo una solución perfecta para aplicaciones sencillas sin efectos colaterales, puesto que reconstruye únicamente aquellos componentes que necesitemos reconstruir, y no todo el árbol de componentes, haciendo que el rendimiento mejor de manera significativa.

El funcionamiento es una mezcla de los dos anteriores, donde en este caso tenemos dos clases.

En primer lugar, tenemos el **StatesRebuilder**, que será la clase de la cual extiendan nuestra lógica para crear nuestro propio modelo, sin necesidad de crear una clase que sirva de modelo, como sucede con Scoped Model y BLoC.

En segundo y último lugar, tenemos el **StateBuilder**, que servirá para encapsular dentro del constructor todos los componentes que queremos tener controlados, para que en cualquier momento podamos reconstruir dicho componente, pudiendo de esta forma reconstruir aquellos componentes que queramos en cada momento y de forma totalmente dinámica.

Podemos entonces definir las siguientes ventajas:

- Lógica, interfaces y control de estado separados.
- Podemos reconstruir únicamente los componentes que necesitemos que cambien de estado y no todo el árbol de componentes, aumentando el rendimiento de la aplicación significativamente.
- Fácil de entender y de utilizar.
- La complejidad de la aplicación no afecta a la complejidad del almacén de estados.

En cuanto a las desventajas tenemos las siguientes:

- Es una liberaría de terceros, lo cual significa que podría dejar de ser mantenida en cualquier momento.
- No es muy conocido hoy en día y por tanto pueden existir errores que no estén contemplados, además de que es complicado encontrarlo.
- Debes tener conocimientos mínimos tanto de Scoped Model, como de BLoC, ya que es una mezcla de ambos.

En nuestro caso, únicamente tuvimos que elegir entre Scoped Model y BLoC, puesto que en el momento de desarrollo de la aplicación States Rebuilder aún estaba en desarrollo, y pese a ser la mejor opción, decidimos no incluirlo entre las posibilidades, puesto que no podíamos saber cómo evolucionaría y que se convertiría hoy día en un patrón/plugin muy utilizado.

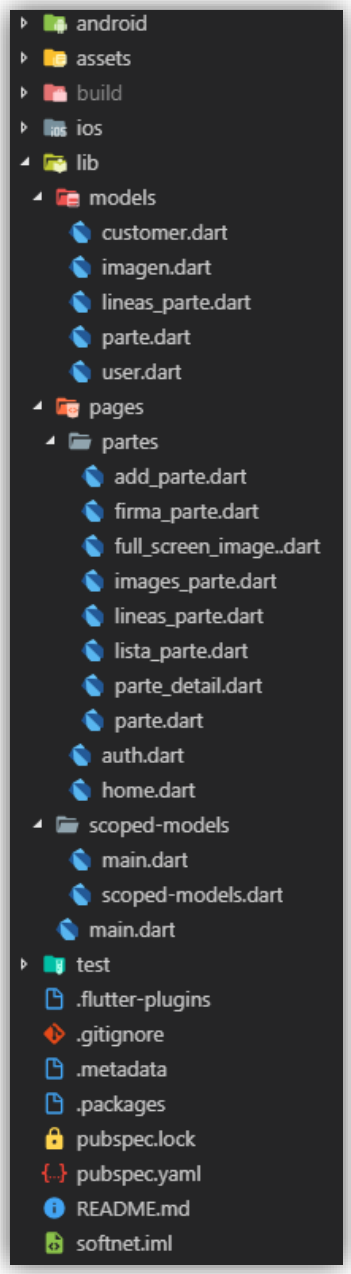


Ilustración 28 - Estructura del proyecto

Finalmente, hemos escogido Scoped Model, puesto que la aplicación a desarrollar no contiene mucha lógica detrás y no contiene interfaces complejas donde intervengan muchos factores externos en ellas y por tanto no hay que controlar de manera exhaustiva el estado en cada momento. Además, nos resultó muy cómodo de utilizar y comprender, por lo que creemos que para una aplicación del tamaño y complejidad que nos ocupa en el proyecto, era la mejor opción.

Para la utilización del patrón, hemos seguido al pie de la letra las instrucciones, consejos y buenas prácticas de los creadores, por lo que creamos una carpeta que contendría todos los modelos que posteriormente gestionaríamos en cada widget.

Además, siguiendo las buenas prácticas, pusimos prácticamente toda la lógica de negocio dentro de este fichero, que a grandes rasgos es el que se encarga de llevar a cabo las llamadas a la API, manejar los datos devueltos por la llamada y devolvérselos a la interfaz para que los pintara. La estructura del proyecto quedaría de la siguiente manera.

Como podemos observar en la imagen, disponemos de una carpeta llamada scoped-models, donde tenemos la inmensa mayoría de lógica y también todas las llamadas a la API, para de esta forma tener un control total del estado de la aplicación según estemos en un componente u otro, pudiendo reconstruir el árbol en cualquier momento, por ejemplo, al pulsar un botón que necesite realizar una petición a la API.

Dentro del archivo *scoped-models.dart*, tenemos todas las llamadas a la API y toda la lógica de control de la aplicación. Vamos a ver un pequeño ejemplo donde en una interfaz de la aplicación necesitamos disponer de la lista de todos los clientes disponibles para posteriormente mostrarlos en un desplegable y escoger uno de ellos para cumplimentar un formulario de creación de un parte.

```dart
Future<Null> getCustomers() async {
  final String url = urlBase + customerUrl;
  final headers = {
    'Contet-Type': 'application/json',
    'Authorization': 'Bearer ' + _authenticatedUser.token
  };

  _isLoading = true;
  notifyListeners();

  http.Response response = await http.get(url, headers: headers);

  final Map<String, dynamic> responseData = json.decode(response.body);
  final List<Customer> listaCustomers = [];

  if (!responseData.containsValue('error')) {
    responseData['customers'].forEach((dynamic customer) {
      final Customer _customer = Customer(
          id: int.parse(customer['id']),
          name: customer['nombre'],
          address: customer['direccion'],
          city: customer['poblacion'],
          cp: int.parse(customer['cp']),
          province: customer['provincia'],
          cif: customer['cif'],
          code: customer['codigo']);

      listaCustomers.add(_customer);
    });
    _customers = listaCustomers;
  }
}
```

Ilustración 29 - Fragmento de código de llamada a la API

En primer lugar, vamos a ver la llamada a la API, donde podemos observar que es una petición asíncrona para mantener en marcha el ciclo de vida y haciendo uso de las funcionalidades que nos proporciona Scoped Model para avisar a la interfaz de si se están llevando a cabo peticiones de manera asíncrona o no.

Si observamos detenidamente, asignamos el valor true a la variable *_isLoading*, que es una variable almacenada que indica si se están realizando a cabo trabajos en segundo plano y por tanto tiene que mostrar por pantalla una barra de proceso, que en nuestro caso es un círculo que tiene animación, y posteriormente hacemos una llamada a *notifyListeners()* que provocará que el árbol de componentes desde el padre hasta cada uno de los hijos se reconstruyan y por tanto según el estado de las variables que tengamos almacenadas, realizarán unas acciones u otras.

La llamada a dicho método se realiza al inicio del ciclo de vida del componente que corresponde a la interfaz del formulario, haciéndolo de la misma manera que realizaríamos promesas en cualquier otro lenguaje de programación, es decir, realizando operaciones cuando la operación asíncrona se complete.

```
@override
initState() {
  super.initState();
  widget.model.getCustomers();
  widget.model.getAlbaranNumber().then((int numero) {
    _lastAlbaranNumber = numero.toString();
  });
}
```

Ilustración 30 - Fragmento de código de inicialización de un widget

Entonces, al finalizar, volveremos a notificar que se ha terminado de cargar los datos, para que se reconstruya y esta vez puesto que no está cargando datos, mostrará el formulario con el desplegable de clientes disponible.

Para finalizar, cabe destacar que utilizando este patrón ha sido significativamente más fácil que usando el *vanilla* que nos proporcionaba Flutter en primera instancia y que realizamos un par de aplicaciones sencillas con ello. Además, hemos podido diferenciar la lógica de las llamadas a los servicios y de las interfaces, haciendo que sea una aplicación muy fluida.

LIBRERÍAS Y PLUGINS UTILIZADOS

Vamos a ver de manera breve todas las librerías de terceros que hemos utilizado en el desarrollo de la aplicación móvil, los cuales Flutter nos proporciona a través de su plataforma Pub.dev[41], donde podemos encontrar las librerías y su puntuación según distintos factores, como la popularidad, el mantenimiento, si está actualizado, etc, la cual nos proporciona una puntuación por cada librería.

Además son fáciles de incluir en el proyecto, ya que únicamente hay que incluirlos en el fichero *pubspec.yaml* donde van todas las referencias tanto a librerías como a imágenes o recursos externos.

La lista de las distintas librerías son las siguientes:

- **Scoped model:** utilizado como patrón y control de estados.
- **Http:** utilizado para gestionar funciones de alto nivel que facilitan el consumo de recursos http y https.
- **Shared preferences[42]:** utilizado para guardar en memoria el token y el usuario que inicia sesión, para poder inyectar el token en las peticiones futuras y poder comprobar la caducidad del token. Además, necesitamos el usuario en algunas de las peticiones e interfaces para mostrar el nombre del técnico y su número de identificación.
- **Curved navigation bar[43]:** utilizado para implementar de manera sencilla una barra de navegación animada, que de un efecto más logrado al cambio de los distintos *tabs*.
- **Path provider[44]:** necesario para poder obtener la localización del sistema de archivos del sistema operativo, para poder acceder a este y poder, por ejemplo, crear un fichero y guardarlo en el dispositivo.
- **Material design icons flutter[45]:** necesario para poder utilizar los iconos basados en *Material Design[46],* ya que dispone de muchos más iconos que los que vienen por defecto.
- **RFlutter alert[47]:** utilizado para mostrar *popups* totalmente personalizables y mucho más conseguidos que los originales.
- **Mime-type[48]:** necesario para obtener la ruta de ficheros y el nombre de estos, además del tipo de fichero, para poder obtener imágenes y crearlas en el dispositivo.

- **Photo view[49]:** utilizado para poder manipular imágenes de la aplicación, permitiendo hacer *zoom*, rotarlas o moverlas, para que ofrezca más flexibilidad al usuario a la hora de visualizar las fotografías realizadas.
- **Date time picker formfield[50]:** es un widget utilizado para facilitar la utilización de selectores de fecha más personalizados.
- **Folding cell[51]:** es un widget utilizado para crear un *container* que sea dinámico y animado de manera muy sencilla y pudiendo personalizar todo su contenido.
- **Flutter signature pad[52]:** necesario para que el usuario pueda firmar en el dispositivo y exportarlo como una imagen.
- **Flutter Advanced Network Image Provider[53]:** utilizado para poder utilizar las imágenes *cacheadas* y poder volver a utilizarlas sin tener que volver a descargarte el contenido.
- **Image picker[54]:** plugin utilizado para poder acceder al sistema de archivos para escoger una imagen o para realizar una fotografía en el momento, pudiendo ser guardadas en el dispositivo.

RESULTADOS VS. MOCKUPS

En este apartado vamos a comparar los bocetos que se han presentado al cliente, con el resultado final de todas las pantallas y vamos a ver la evolución o cualquier cambio que se haya producido por la entrega continua y las grandes posibilidades que nos ofrecen las librerías de terceros.

Además, hablaremos también sobre qué apartados estéticos y visuales provenientes de librerías externas y de las razones por las cuales se han escogido los colores que representan la aplicación.

Vamos a ver cada apartado individualmente, para que podamos comprobar los resultados finales del desarrollo de la aplicación móvil.

INICIO DE SESIÓN

Ilustración 32 - Mockup login

Ilustración 31 - Inicio de sesión

Esta es la primera interfaz que los usuarios verán al acceder a la aplicación y a la cual volverán en caso de cerrar sesión o que el token expire y por tanto requerirá de un nuevo inicio de sesión para refrescarlo y poder realizar peticiones a la API.

Como podemos observar la interfaz de inicio de sesión prácticamente no ha cambiado, salvo obviamente el diseño y el estilo de la interfaz. Para los colores de la aplicación, hemos optado por elegir tres colores

que sean triádicos[55] a partir del color principal de la aplicación, de los cuales hemos usado uno de ellos para los botones y para acentuar algunas partes de la aplicación, ya que este tipo de colores funcionan muy bien para acentuar el principal.

Ilustración 35 - Mockup lista vacía

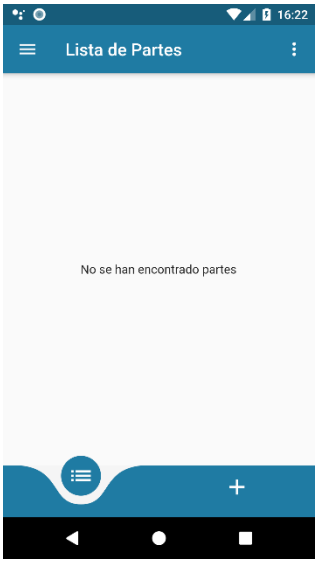


Ilustración 34 – Lista de partes vacía

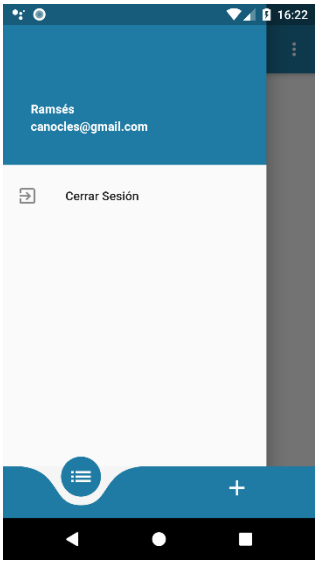


Ilustración 33 – Menú emergente

LISTA DE TAREAS

En este caso, podemos observar cómo hemos mantenido la disposición original de los elementos, pero durante el desarrollo se han cambiado algunos elementos por simple estética, como puede ser el botón de ordenar se ha convertido en el icono habitual para abrir un menú en la barra de navegación y la barra de navegación inferior que está construida mediante una librería de terceros que nos permite realizar formas y animaciones más conseguidas. Además, hemos añadido un *drawer* para poder cerrar sesión en la interfaz principal.

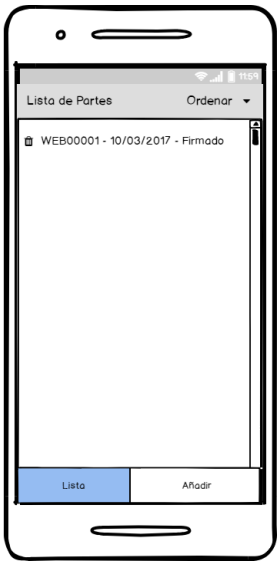


Ilustración 36 - Mockup lista partes

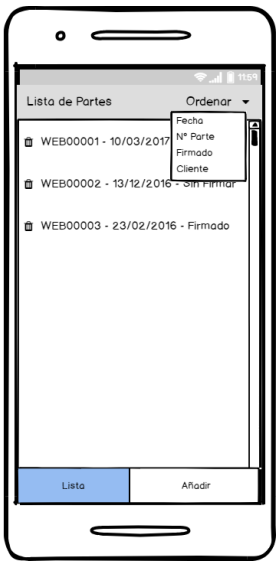


Ilustración 37 - Mockup lista ordenada

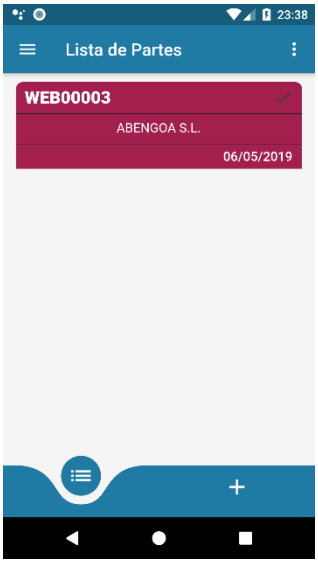


Ilustración 38 - Lista partes

En caso de que tengamos partes, sí que ha habido cambios significativos por razones lógicas, ya que, en los bocetos es complicado plasmar hasta qué punto podemos llegar con el diseño de una aplicación y conforme avanzas en el desarrollo siempre aprendes cosas nuevas que pueden servirte para mejorar tus interfaces. En este caso, los datos mostrados siguen siendo los mismos, salvo que en este caso tenemos el nombre del cliente y un icono de un *check* que nos indica si está firmado o no, según está de color verde o gris respectivamente.

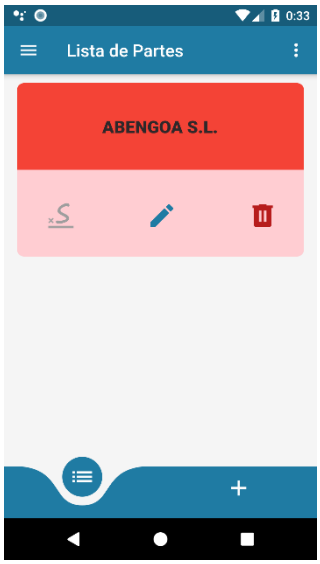


Ilustración 39 - Parte desplegado

Durante el desarrollo hemos descubierto una librería que nos permite hacer una especie de desplegable del propio contenedor a modo de tarjeta, pudiendo personalizar tanto las animaciones como los colores, permitiendo conseguir un resultado muy vistoso, que además nos ayuda a colocar acciones sobre ese mismo parte, sin necesidad de que sea una lista al uso.

Tenemos tres acciones por parte, el primer icono que por su color nos debería indicar que está deshabilitado es para firmar el parte directamente desde la lista, que estará disponible para pulsarlo cuando añadamos alguna línea al parte. En cuanto al segundo icono, en caso de no estar firmado, veremos el icono más habitual para indicar que vamos a modificar dicho parte y en caso de que se haya firmado, el icono cambiara a un icono de un ojo, para denotar que únicamente podremos visualizar los datos, y por último tenemos el icono para borrar el parte.

El icono derecho de la barra de navegación superior, lo utilizamos para escoger la ordenación de la lista de partes, reutilizando los que teníamos en el boceto, puesto que son los más utilizados, ordenando de manera ascendente la primera vez que pulsamos sobre una de las propiedades por las que ordenar o filtrar y en caso de volver a pulsar reordenaremos de manera descendente.

En el caso de querer filtrar por firmado o no firmado, no ordenará, si no que filtrará la lista existente mostrando únicamente los elementos que cumplan con la condición según pulsemos por primera o segunda vez.

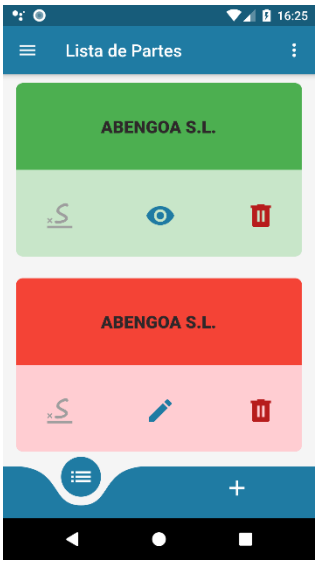


Ilustración 41 - Parte firmado y no firmado

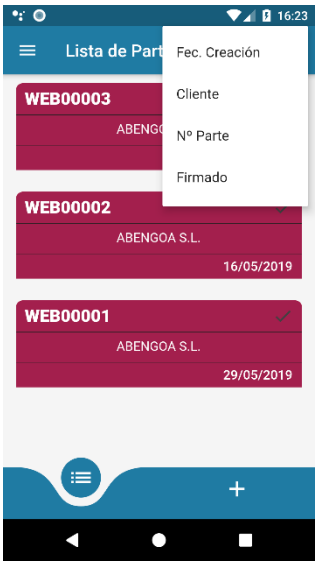


Ilustración 40 - Lista de partes ordenados

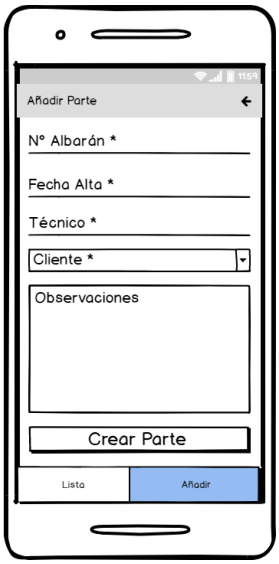


Ilustración 43 - Mockup creación de parte

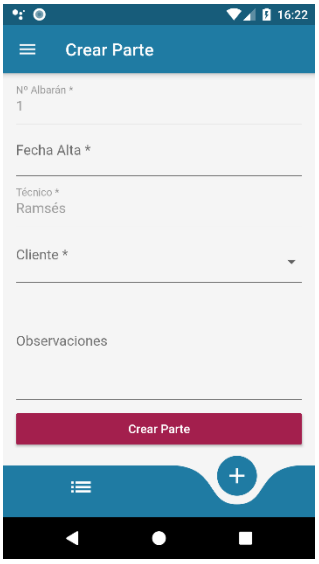


Ilustración 42 - Creación de parte

AÑADIR PARTE

Como también sucede con el inicio de sesión, esta interfaz también se ha mantenido fiel al boceto, ya que no ocupa mucho en la pantalla y no se han requerido más datos para realizar la inserción de un parte nuevo en la base de datos.

Como es de suponer, al pulsar el botón antes de realizar la petición pasa por una validación de todos los campos para comprobar que se están cumplimentando correctamente todos los campos. En este caso, tanto el número de albarán como el técnico están deshabilitados, ya que ambos vienen predeterminados por el próximo número disponible de parte de la base de datos y por el técnico que ha iniciado sesión.

Además, tanto la fecha de alta como el cliente son campos obligatorios, por lo que en caso de no rellenarlos se nos marcaran en rojo y aparecerá un mensaje debajo del campo indicando que ha provocado el fallo.

Una vez realizada la consulta y obteniendo una respuesta positiva por parte de la API, indicando que se ha insertado con éxito el parte, nos redirigirá a la lista de partes con el parte añadido, pero en caso contrario aparecerá un mensaje de error que nos indicará que ha podido pasar.

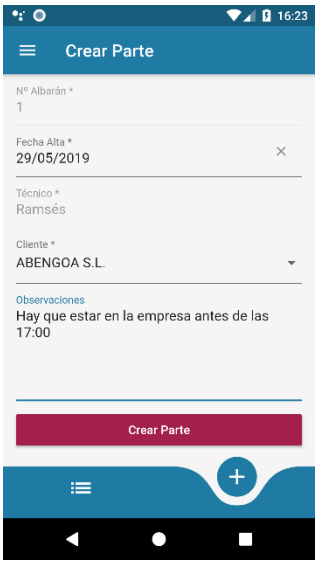


Ilustración 44 - Creación de parte

DATOS DEL PARTE

Ilustración 46 - Mockup de los datos de un parte

Ilustración 45 - Datos del parte

También en la interfaz de detalles de un parte concreto, se ha mantenido tanto la barra de navegación superior, como la sección de datos principales del parte, que únicamente es para visualizar datos, y por tanto no se podrá modificar en ningún caso.

Además, que hemos añadido iconos para añadir mejoras estéticas que ayudan al flujo de trabajo que realiza el usuario.

LÍNEAS DEL PARTE

Ilustración 49 - Mockup líneas de parte vacías

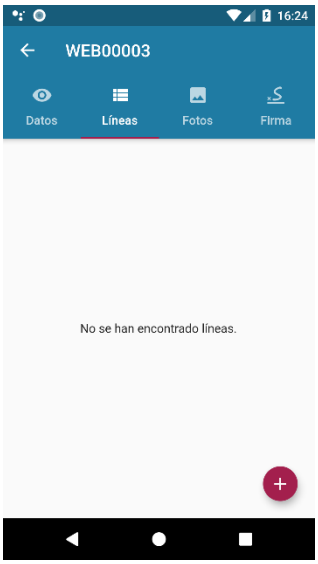


Ilustración 48 - Líneas de parte vacías

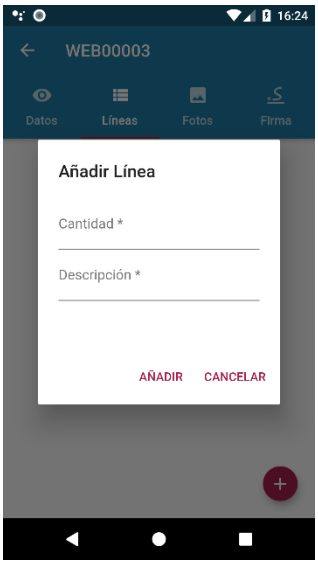


Ilustración 47 - Añadir línea

Como podemos observar, esta sección del detalle del parte también se ha mantenido igual al boceto, salvo que en un principio había una interfaz a parte para insertar las líneas, pero conforme avanzaban las iteraciones acordamos que era más sencillo plantear la inserción mediante un *popup* para facilitar el flujo y ahorrarnos interfaces innecesarias.

Ilustración 50 - Mockup líneas de parte

Al igual que pasa con todos los formularios antes de realizar la consulta pasan por una serie de validaciones, en caso de que todo haya ido bien se cerrará el *popup* y veremos la línea insertado, y en caso contrario aparecerá un mensaje de error informándonos de que ha pasado.

Como podemos observar se ha mantenido la estructura del boceto inicial, pero se ha añadido un icono que representa el borrado de la línea en caso de querer eliminarla. Lógicamente también nos preguntará antes de eliminar la línea para confirmar el borrado o cancelar la operación.

Una vez hayamos incluido una línea, podremos introducir la firma del cliente para dar por terminado el parte. Pero hay que tener en cuenta que una vez firmado ya no podremos realizar ninguna inserción o modificación al parte, así que hay que asegurarse de que se ha insertado todo lo necesario.

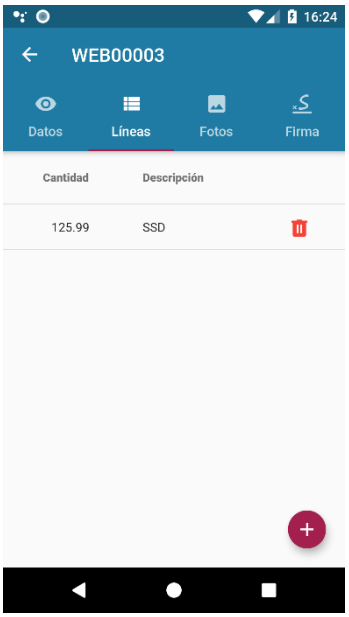
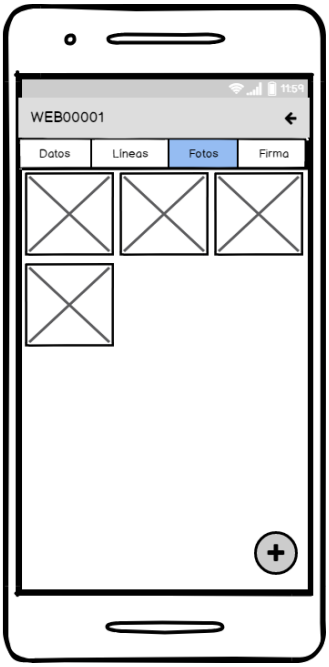


Ilustración 51 - Líneas de parte

FOTOS DEL PARTE



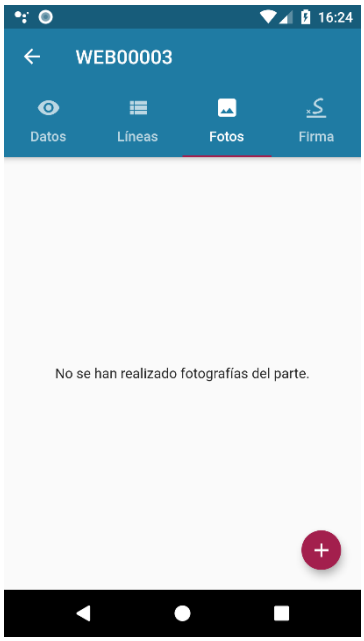


Ilustración 53 – Parte sin fotos

Ilustración 54 – Escoger fotografía

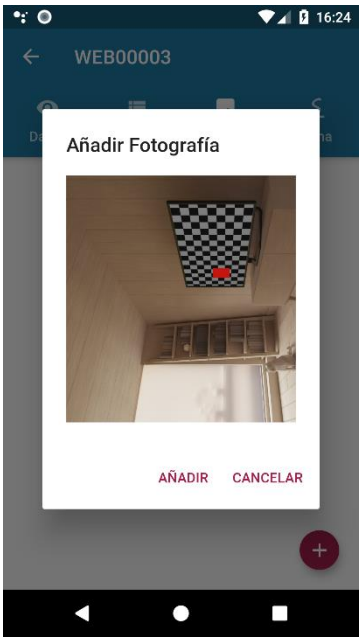


Ilustración 52 – Añadir fotografía

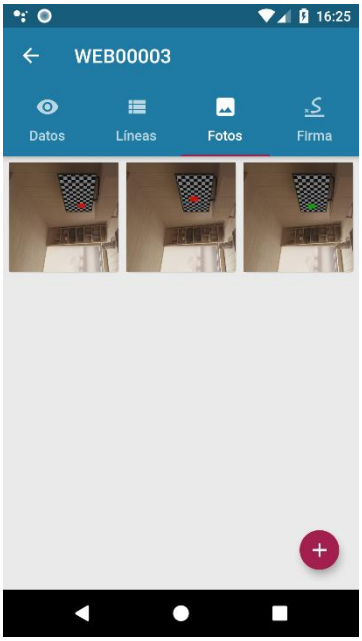


Ilustración 56 - Lista de fotos del parte

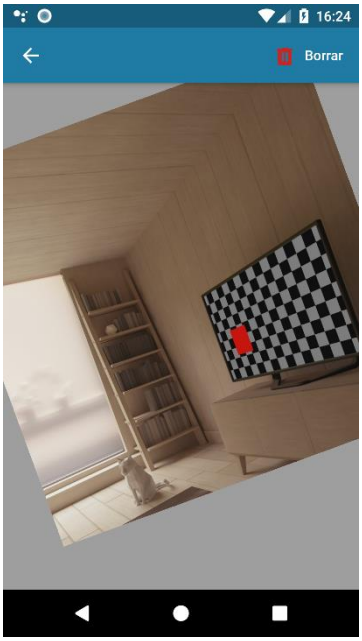


Ilustración 55 – Detalle de la foto

En este caso, es exactamente igual que el boceto, debido a que, en este caso concreto, ya habíamos realizado pruebas con aplicaciones anteriores en los cursos y en ejemplos y sabíamos cómo era el componente que nos ofrecía la librería, por lo que es lógico que se parezcan tanto.

Salvando la diferencia, de que escogimos de nuevo el *popup* como componente para previsualizar la imagen escogida o realizada, añadirla o no al parte, además, todas las imágenes las renderiza a partir de una URL, una vez almacenadas en el servidor por lo que en caso de que no se puedan recuperar se mostrará una imagen *dummy* para que no aparezca un error que escandalice al usuario.

Incluso podemos ver la imagen con más detalle pulsando sobre ella y pudiendo en caso de requerirlo borrarla o manipularla libremente, tanto para hacer *zoom* como para rotarla o moverla en el espacio de la pantalla.

FIRMA DEL PARTE

Ilustración 58 – Mockup de parte sin firmar

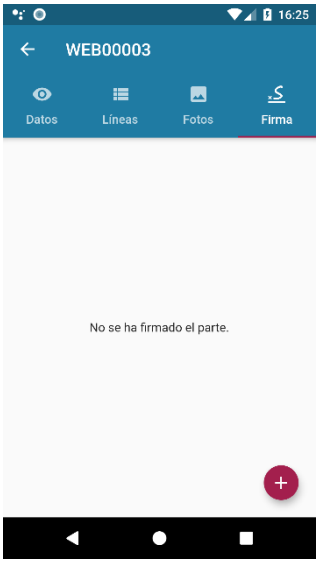


Ilustración 59 – Parte sin firmar

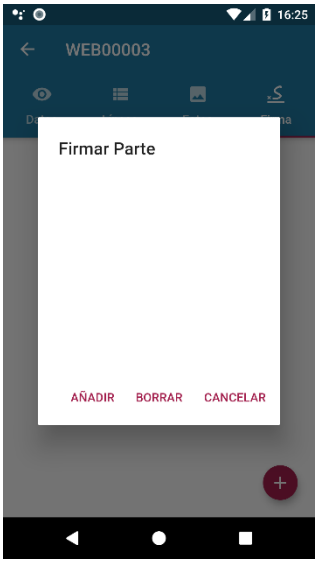


Ilustración 57 – Firmar parte

En la sección de la firma tenemos el mismo caso que los anteriores. Mantenemos la misma estructura y la misma información, pero al igual que con las líneas y las fotos, decidimos firmar en un *popup* para poder borrar la firma en caso de que no nos guste y añadirla en caso de que estemos satisfechos, ya que, en muchas ocasiones no estamos acostumbrados a firmar en dispositivos y las firmas no salen como nos gustaría.

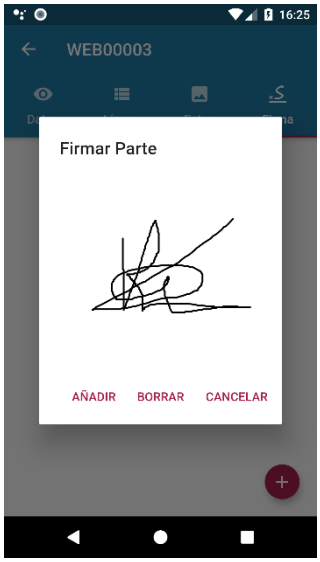


Ilustración 61 – Firma del parte

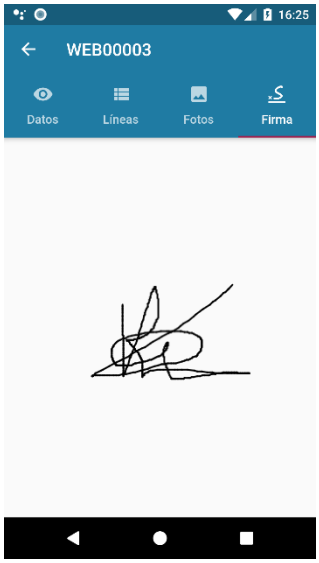


Ilustración 60 – Parte firmado

Para añadirla pulsaremos sobre añadir, y si todo ha ido bien, se cerrará el *popup* y veremos la firma en la sección, la cual podremos ampliar y manipular a nuestro antojo, pero si observamos detenidamente, habrá desaparecido el botón flotante para añadir la firma y además también habrá sucedido lo mismo en todas las demás secciones en el detalle del parte, bloqueando cualquier inserción, modificación o borrado de elementos relacionados con el propio parte.

En el caso de los borrados, como puede ser el de líneas y fotos, habrán desaparecido.

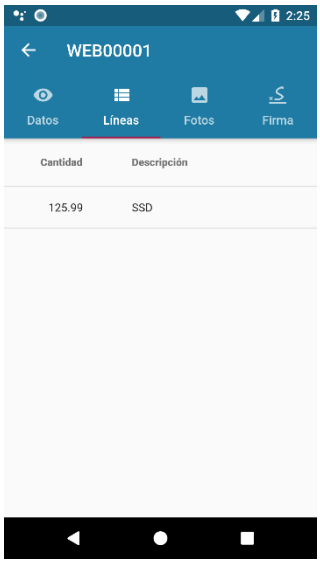


Ilustración 62 – Líneas bloqueadas

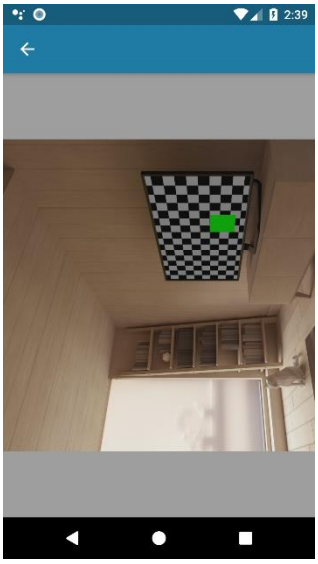


Ilustración 64 – Foto bloqueada

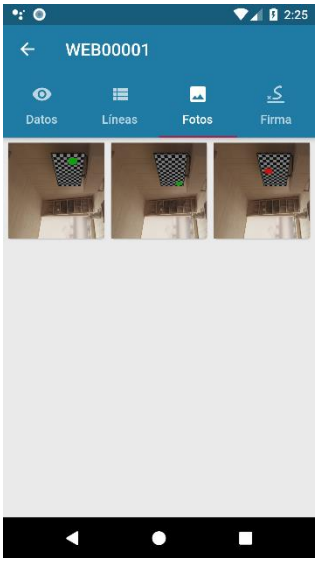


Ilustración 63 – Fotos bloqueadas

PRUEBAS

En este apartado vamos a ver algunas de las pruebas que hemos realizado a lo largo del desarrollo para de esta forma poder cerciorarnos de que el proceso de desarrollo no se ve comprometido o retrasado por errores que podrían haberse evitado.

API REST

Para probar la API, hemos optado por utilizar Postman[56], que es una aplicación que nos permite realizar consultas HTTP y HTTPS a cualquier URL, lo cual nos ayudará a saber si los *endpoints* que vamos implementando están funcionando correctamente y devuelven los resultados que hemos previsto en cada uno de los casos.

Para realizar las pruebas de manera más o menos automática, sin tener que estar cambiando entre las URLs de producción y desarrollo o para recoger los tokens sin necesidad de estar copiando y pegando según expiraba o necesitábamos probar que no funcionaba creamos varios entornos con variables definidas en ellos, los cuales nos permitían mediante scripts recoger de manera dinámica.

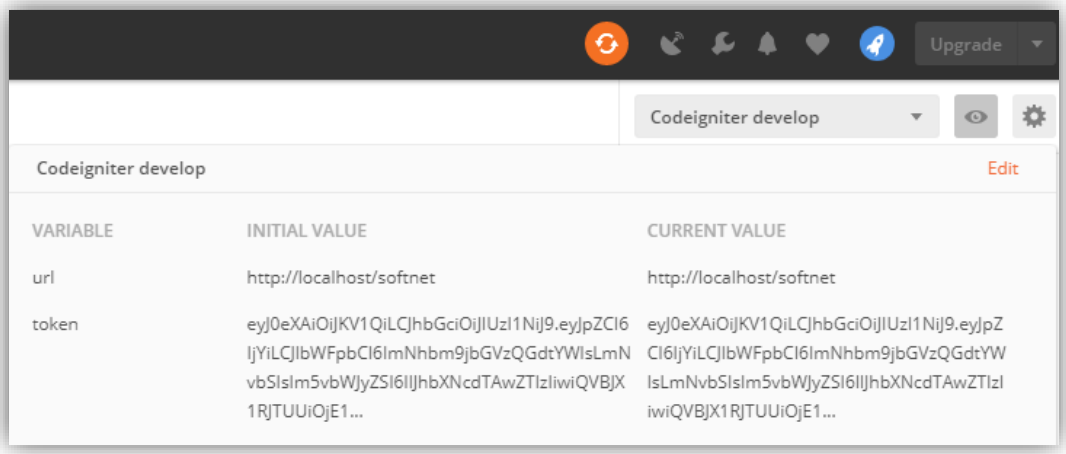


Ilustración 65 – Entornos de Postman

Podemos ver como en el entorno de desarrollo tenemos dos variables, url y token, donde la primera corresponde a la url base del proyecto, la cual, nos ayudará a que en caso de necesitar cambiar la url, únicamente tengamos que hacerlo una vez y la segunda un token que recogerá dinámicamente mediante el script que podemos ver a continuación.

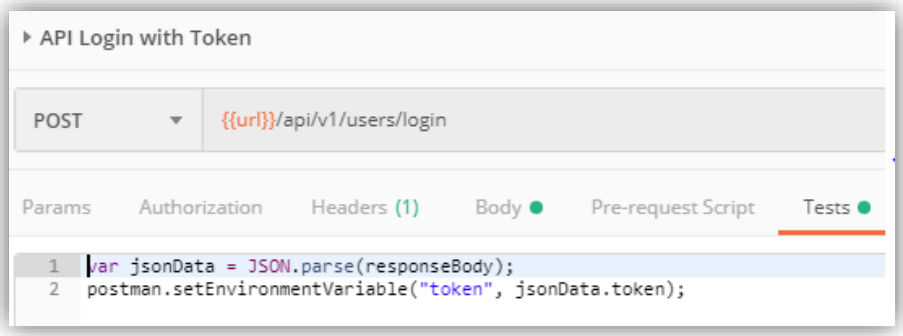


Ilustración 66 – Script para automatizar la recogida de tokens y aplicación de estos

Dicho script está definido en el apartado de tests, la cual se ejecuta cada vez que realizamos la petición. Cuando realiza la petición y nos responde la API, cogemos la variable token de la respuesta y la asignamos a la variable de entorno, la cual utilizamos en las cabeceras de todas las peticiones siguientes.

Si observamos detenidamente la url, podemos observar que hay una parte en naranja la cual está encapsulada por dos corchetes, donde estos corchetes nos indican que estamos usando una variable de entorno o global.

Vamos a ver un ejemplo concreto, para que sea más fácil de entender el proceso de prueba y como utilizamos la variable de entorno token de manera dinámica.

Tenemos que recordar que todas las rutas comparten una URL Base que es la siguiente:

- http://localhost/softnet/api/v1

Como ejemplo, hemos escogido una de las rutas más usadas en la aplicación, que es la obtención de los partes del técnico. Para ello, observemos la siguiente captura.

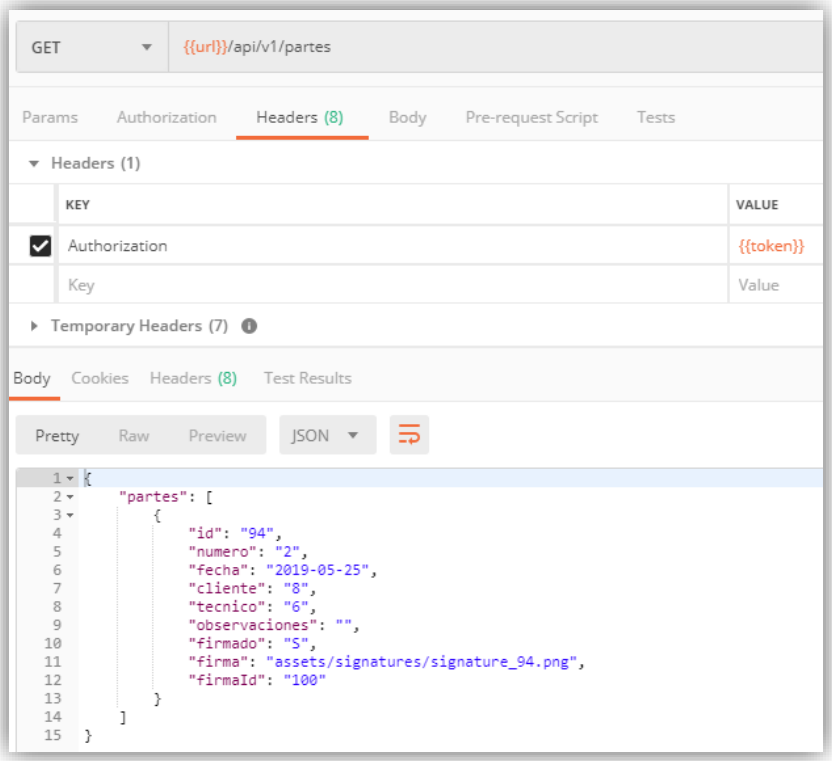


Ilustración 67 – Consulta exitosa con lista de partes del usuario

Como podemos observar en la cabecera, le estamos pasando el token que anteriormente hemos obtenido a partir del *login* de manera automática tras recibir la respuesta.

En caso de que se haya introducido la cabecera, el token sea correcto y no haya expirado, entonces nos devolverá la lista de partes del técnico mediante el token que estará asociado a dicho usuario

Pero también está la posibilidad de que ocurra cualquier tipo de error, los cuales vamos a tratar de reproducir y discutir.

Pongamos el caso de que el token no ha sido proporcionado y, por tanto, no debería tan siquiera llegar a realizar ninguna petición a la base de datos, puesto que es la primera validación de cualquier ruta. La respuesta es la siguiente:

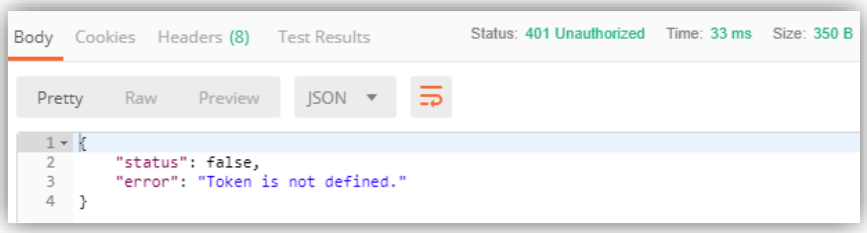


Ilustración 68 – No se ha proporcionado token

Como podemos observar, obtenemos un 401 que nos indica que no estamos autorizados a realizar peticiones a esa ruta concreta.

También tendríamos el caso en el cual el token ha expirado y tampoco debería dejarnos realizar la petición, obteniendo el mismo resultado anterior, salvando el mensaje que debería ser distinto.

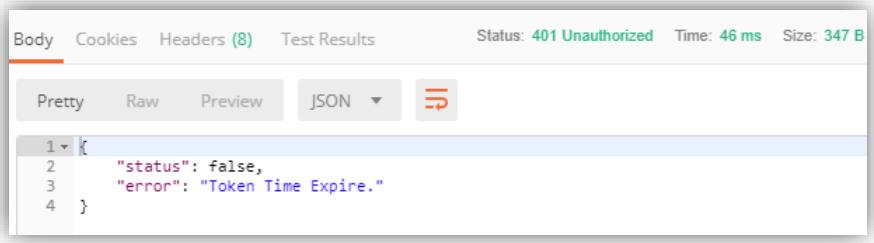


Ilustración 69 – Respuesta de token caducado

Efectivamente, al realizar la petición obtenemos el mismo código de estado, pero esta vez nos indica que el token ha expirado, pero ¿qué pasaría si intentamos manipular el token?

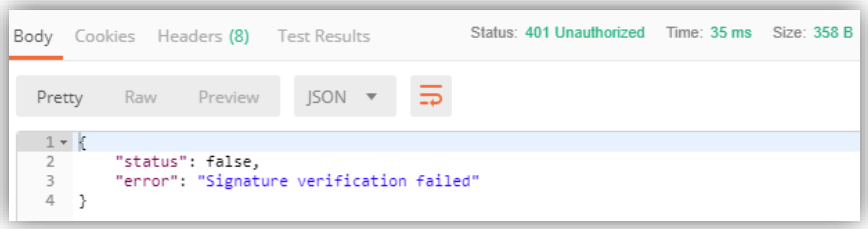


Ilustración 70 – Respuesta de token manipulado

Lógicamente, en este último caso de error en el token, tampoco podrías acceder a la información de esa ruta y nos advertiría del error.

Además, la ruta que estamos tomando como ejemplo, también tiene la posibilidad de que no encuentre ningún parte del técnico asociado al token y por tanto nos tendrá que advertir de ello. En nuestro caso, hemos optado por definir una serie de mensajes de error claros, que sean fácil de reconocer en el cliente y que equivalgan a una cadena explicativa del error en la API.

Ilustración 71 – Respuesta con la lista de partes vacía

INTEGRACIÓN E IMPLANTACIÓN

En este apartado vamos a hablar sobre el proceso de integración del nuevo sistema desarrollado con el sistema vigente.

Para empezar, se han presentado los cambios necesarios en la parte común a ambos sistemas, es decir, la base de datos, la cual es compartida tanto por la API como por el servidor web que contiene la aplicación web.

Los cambios que hemos propuesto a la empresa son los anteriormente vistos en el apartado de análisis, donde hemos visto el modelo E-R, antes de aplicarlos se han discutido y justificado, puesto que es importante que los cambios que se vayan a realizar en la base de datos no provoque ningún fallo en la misma, ya que el sistema original ya está en producción y conllevaría una parada de los servicios proporcionados por la empresa, lo cual sería un problema grave.

La parte positiva es que tanto las tablas como las claves ajenas aceptan nulos, por lo que introducir todos estos elementos nuevos no conlleva ningún problema de integración referencial ni de índices.

El siguiente paso para la integración, se le ha proporcionado la API al técnico de la empresa, para que prepare el proyecto en uno de sus servidores, dándonos acceso a su dominio y por tanto a la API para poder trabajar de manera remota y que la aplicación móvil funcionase sin necesidad de tener que estar en la misma red que el sistema. Una vez preparado el servidor con acceso a la API, se han realizado cambios en la configuración de la API con respecto a la base de datos, para que la API se conecte con una base de datos replicada de la original para la realización de pruebas.

Tras tener el servidor en marcha y acceso a la API, todo ello probado con Postman para asegurarnos de que todas las rutas estaban funcionando como en el entorno de desarrollo, generamos el *apk* de la aplicación y se la instalamos en un par de dispositivos para que se pudieran realizar pruebas desde distintos dispositivos con distintos técnicos realizando peticiones concurrentes, poniendo a prueba el nuevo sistema ya integrado con el original.

Una vez realizadas las pruebas necesarias con distintos dispositivos, cambiamos la configuración de la API para que se conecte con la base de datos en producción, para de esta manera, el cliente web pueda visualizar los partes que se van creando a partir de la aplicación móvil y de esta manera poder ver los partes realizados en el sistema nuevo, desde el sistema original.

PROBLEMAS ENCONTRADOS

En este apartado hablaremos sobre los problemas que hemos encontrado a lo largo del desarrollo, implementación e integración del proyecto.

Para empezar, tras decidir el framework sobre el que íbamos a trabajar, basándonos en los artículos, comunidades y demás factores que ya hemos explicado en el apartado de estudio de mercado, la primera barrera que nos hemos encontrado es la sensación de estar aprendiendo algo que será posiblemente uno de los frameworks mejor valorados para desarrollar aplicaciones móviles, pero al mismo tiempo, de estar trabajando con un framework aún en fase de descubrimiento y desarrollo, el cual nos ha dado la sensación en el inicio de ser susceptible a gran cantidad de errores e incompatibilidades por el uso de una gran cantidad de librerías de terceros.

Sin embargo, conforme hemos ido aprendiendo mediante cursos, ejemplos y práctica, esa sensación desaparecía por completo para convertirse en una total satisfacción a la hora de realizar algunas interfaces y funcionalidades.

Pero, de nuevo, tras superar la primera curva de aprendizaje, nos encontramos con el problema de los patrones y de cómo comunicar todas las interfaces entre sí sin que fuese una pesadilla tener que pasar a modo de parámetros los estados de las anteriores. Por lo que aprender cómo funcionaba el ciclo de vida de Flutter según el patrón fue otro reto que se ha tenido que afrontar y superar para poder escoger un patrón y la mejor manera que había de solucionar ese mismo problema en aquel momento.

También otro de los problemas encontrados era la necesidad de tener que realizar la API en Codeigniter, y la adaptación de su sistema a los nuevos cambios introducidos tanto en la base de datos como en el cliente web, los cuales aún no se ha podido poner en producción por temas logísticos de la propia empresa, la cual no ha tenido tiempo de integrar ni tampoco de probar, pero que esperamos que en breve podamos poner en funcionamiento para así comprobar que todo el trabajo realizado funcione igual de bien que nos funciona en desarrollo.

Para finalizar, otro problema que en sí mismo también puede ser una ventaja, pero que en el caso de nuestra poca experiencia con las tecnologías fue más bien la primera opción, era la maleabilidad y constante evolución del *framework* proveniente de su muy reciente publicación para ser utilizado, lo cual hacía que en ocasiones nos encontrásemos errores que provenían del propio *framework* y que hasta pasadas unas semanas no resolvían en las *issues* de la propia plataforma.

AMPLIACIONES Y MEJORAS

En cuanto a las ampliaciones y mejores, tras toda la experiencia y práctica obtenida durante la realización del proyecto, hemos obtenido ideas y conocimientos que posiblemente mejoren en gran medida la aplicación y el uso de esta en el contexto y entorno de la empresa que la utilizará.

En primer lugar, utilizaríamos una plataforma muy útil para integración continua[58] y entrega continua[59], la cual hemos descubierto posteriormente a terminar el proyecto. La plataforma se llama Bitrise[57] y nos proporciona un entorno similar a Travis[60] o Jenkins[61], los cuales podemos utilizar para que cada vez que mezclemos con la rama principal, por ejemplo, pase las pruebas de Flutter, al igual que las pruebas de estrés, cobertura y posibles bajadas de rendimiento en ciertos componentes. Y lo más interesante de esta plataforma es que puedes probar en distintos entornos para saber si tanto en Android como en iOS está funcionando cada componente por separado.

En segundo lugar, utilizaríamos una combinación de dos librerías que han mejorado significativamente estos últimos meses y que nos brindan la posibilidad de comprobar la conexión del dispositivo, tanto si estamos conectados por Wifi o por los datos del dispositivo, para que en caso de no tener conexión avisar al usuario para que cuando realice peticiones a la API no obtengamos conexiones rehusadas, que pese a estar controladas en la aplicación, no podemos preverlas ni tampoco anticiparnos a la petición sin conexión. Además, desarrollaríamos un sistema de reintentos, mejorando la tolerancia de errores en cuanto a peticiones a la API, para que en caso de que nos hayamos quedado sin conexión a la hora de crear un parte, lo guarde en memoria, para reintentarlo una vez vuelva a tener conexión. La mejora conllevaría tener que instalar una base de datos SQLite[58] para almacenar de manera temporal todos los elementos que no se han podido enviar al servidor y una vez enviados borrarlos del dispositivo.

Por lo tanto, estas mejoras nos permitirían poder trabajar sin conexión y reduciendo significativamente los errores de conexiones a la API, ya que almacenaríamos todos los datos que no se pudiesen enviar a la API y con el sistema de reintentos podríamos conseguir sincronización desatendida con la base de datos, puesto que el sistema reintentaría realizar las peticiones que han fracasado una vez detecte que tiene conexión a internet.

Otra de las mejoras que hemos propuesto a la empresa es la posibilidad de que cuando un cliente firme un parte, podamos tener la opción de generar un PDF en la API y enviarlo directamente por correo al cliente, quitando la necesidad de tener que imprimirlo en la oficina y dárselo en la próxima visita del cliente en caso de que requiera la información del parte. Dicha mejora conllevaría tener que añadir un campo email en la tabla de clientes, y además recolectar dichos emails para enviar los PDF de manera automática.

Para finalizar, también sería buena idea poder controlar la posición de los técnicos en el momento de abrir y cerrar partes, pudiendo de esta manera tener un mejor control de los tiempos y trabajos que se están realizando en todo momento y, además, aprovechar esta nueva mejora para utilizar mapas para localizar las direcciones de los clientes a los cuales los técnicos tienen que ofrecer servicios, ya que con la integración de la aplicación se podría reinventar la forma de trabajar de la empresa y poder de esta forma planificar las visitas a los clientes, pudiendo asignar partes ya creados a los técnicos que están realizando servicios en tiempo real.

Con todo esto, en nuestra opinión sería una aplicación más que completa y que aportaría más valor a la empresa y les serviría para mejorar su sistema, tanto tecnológico como de trabajo en general. Pero obviamente dichas mejoras pueden no ser una necesidad para la empresa y prescindiendo de ellas, seguirían teniendo una aplicación que cubre sus necesidades actuales y que ayudará en gran medida a

reducir el esfuerzo humano en cuanto a tareas de gestión de partes, pudiendo focalizar dicho esfuerzo en tareas que den un mayor beneficio.

CONCLUSIONES

Tras finalizar con la implementación del proyecto, hemos realizado un ejercicio de retrospectiva, de la cual podemos concluir con que se han conseguido los objetivos propuestos con la empresa, incluso hemos ido un poco más allá, implementando algoritmos de recuperación de número de albarán borrados, para no encontrar huecos en la facturación o también la posibilidad de realizar más de una fotografía o subida de imagen de la galería, aportando más documentación a la realización de un parte o incidencia.

En primer lugar, hemos desarrollado una API REST segura y totalmente funcional, la cual cumple con los estándares definidos para que sea considerada RESTFull, utilizando los métodos HTTP de manera correcta y utilizando los códigos de estado pertinentes en cada situación, tanto de éxito como de error. Además, utilizando la plataforma de Swagger UI[63], hemos documentado detalladamente el funcionamiento del API, lo cual hará más fácil el mantenimiento y la extensibilidad de esta. Además, pese a que la idea inicial de desarrollarla en Codeigniter no nos gustó, ha sido una experiencia enriquecedora, ya que hemos aprendido a utilizar un framework que desconocíamos, lo cual también ha servido para crecer profesionalmente, aumentando los conocimientos de las tecnologías disponibles a nuestro alrededor.

En cuanto a lo que a la elección de las tecnologías para el desarrollo de la aplicación, no podemos estar más contentos, puesto que Flutter y Dart nos han provocado un mayor interés por el framework y el lenguaje, el cual estamos seguros de que queremos utilizar en futuros proyectos, ya sean personales o profesionales. Una de las razones con más peso para decantarnos por Flutter, fue la novedad y que de Ionic ya teníamos una buena base de conocimientos, dado que trabajamos en nuestro trabajo actual regularmente con dicha tecnología, por lo que decidimos utilizar esta gran oportunidad para aprender otra tecnología nueva que nos aporte valor a nuestra carrera como ingenieros informáticos y como profesionales en general.

Por lo tanto, y esperando no dejarme nada en el tintero, estamos muy satisfechos con el trabajo, pese a habernos gustado disponer de más tiempo para adquirir conocimientos más fundamentados y más cimentados antes de realizar una aplicación real, estamos seguros de que hemos aprendido muchísimo con la experiencia y tenemos muy claro que queremos seguir adquiriendo conocimientos de las tecnologías utilizadas. Es más, ya estamos realizando un proyecto personal utilizando Flutter y eso refuerza nuestra convicción de que pese a la solución a la que se ha llegado, no cambiaríamos la tecnología escogida, ya que actualmente nos está proporcionando mucha satisfacción personal, al estar creando desde la nada una aplicación que aporte valor a cualquier persona.

Para finalizar, hemos podido aplicar conocimientos adquiridos en muchas de las asignaturas cursadas durante el grado, como puede ser Seguridad en el Diseño Software, Aplicaciones Distribuidas en Internet, Metodologías Ágiles de Desarrollo del Software y Metodologías y tecnologías de integración de sistemas, y muchas otras que nos han ofrecido conocimientos más que suficientes para ser capaces de llevar a cabo un proceso de desarrollo de software desde los inicios hasta la entrega del este al cliente, junto con el mantenimiento del mismo.

REFERENCIAS Y BIBLIOGRAFÍA

- Apuntes de Metodologías Ágiles y Desarrollo Software
- Apuntes de Seguridad en el Diseño Software
- Apuntes de Aplicaciones Distribuidas en Internet
- Apuntes de Metodologías y Tecnologías de Integración de Sistemas
- Artículos de discusión sobre comparativa de arquitecturas de Flutter
    - Flutter app architecture 101: Vanilla, Scoped Model, BLoC
    - Flutter #OneYearChallenge; Scoped Model vs BloC Pattern vs States Rebuilder
    - Implementa Arquitectura a tu proyecto Flutter usando el patrón BLOC
- Artículos de comparativas entre distintos frameworks para crear aplicaciones móviles, de las cuales hemos podido sacar los datos de las gráficas del apartado de estudio de mercado
    - React Native vs Flutter vs Ionic vs NativeScript vs PWA
    - React Native Vs. Xamarin Vs. Ionic Vs. Flutter: Which Is Best For Cross-Platform Mobile App Development?
    - Flutter Vs Ionic Vs React Native
    - Datos estadísticos del repositorio de Ionic
    - Datos estadísticos del repositorio de flutter
    - Datos estadísticos del repositorio de React native
- Página con muchos ejemplos de Flutter para aprender
- Artículo de discusión sobre las razones por las cuales Flutter está siendo tan utilizado.

[1] Aplicación Web en producción de Softnet: http://www.informaticaenalicante.com/bosoftnet/
[2] Codeigniter, página oficial del framework: https://www.codeigniter.com/
[3] MySQL, página oficial: https://www.mysql.com/
[4] API REST, Application Program Interface, artículo de discusión: https://searchmicroservices.techtarget.com/definition/RESTful-API
[5] Dart, página oficial: https://dart.dev/
[6] Flutter, página oficial: https://flutter.dev/
[7] Scoped Model, librería de terceros utilizada en el desarrollo: https://pub.dev/packages/scoped_model
[8] Git, página oficial y artículo que explica qué es: https://git-scm.com/ y https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/
[9] GitHub, página oficial de la plataforma: https://github.com/
[10] Ionic, página oficial: https://ionicframework.com/
[11] Cordova, página oficial: https://cordova.apache.org/
[12] SDK, Software Development Kit
[13] Aplicaciones híbridas, artículo de explicación: https://solidgeargroup.com/desarrollo-de-apps-hibridas-con-ionic?lang=es
[14] Angular JS, página oficial: https://angular.io/
[15] Compilación con librerías ARM C/C++, artículo de discusión: https://medium.com/@atul.sharma_94062/flutter-how-does-it-works-6e4c73842e67
[16] React Native, página oficial: https://facebook.github.io/react-native/
[17] React, página oficial: https://reactjs.org/
[18] Redux, página oficial: https://es.redux.js.org/
[19] NativeScript, página oficial:
[20] VUE, página oficial del framework: https://vuejs.org/
[21] Metodologías ágiles, manifiesto ágil según los padres del mismo: https://agilemanifesto.org/iso/es/manifesto.html
[22] Scrum, artículo de discusión: https://proyectosagiles.org/que-es-scrum/

[23] Kanban, artículo de explicación y uso: https://kanbantool.com/es/metodologia-kanban
[24] Softnet, página oficial de la empresa: http://www.informaticaenalicante.com/
[25] ERP, definición de un sistema ERP: https://www.ticportal.es/temas/enterprise-resource-planning/que-es-sistema-erp
[26] PYME, qué es según el BOE: http://www.ipyme.org/es-ES/DatosPublicaciones/Paginas/DefinicionPYME.aspx
[27] Modelo E-R, artículo de explicación: https://www.genbeta.com/desarrollo/fundamento-de-las-bases-de-datos-modelo-entidad-relacion
[28] Endpoints, artículo que lo define: https://www.druva.com/blog/simple-definition-endpoint/
[29] Documentación de Flutter, página oficial: https://flutter.dev/docs
[30] Udemy, página oficial: https://www.udemy.com/
[31] JWT, JSON Web Token
[32] JSON, JavaScript Object Notation
[33] Token, definición según Wikipedia: https://es.wikipedia.org/wiki/Token_(inform%C3%A1tica)
[34] Codificación en Base64, definición según Wikipedia: https://es.wikipedia.org/wiki/Base64
[35] RESTFull, definición según MuleSoft, https://www.mulesoft.com/resources/api/what-is-rest-api-design
[36] BLoC Pattern, página de la librería: https://pub.dev/packages/bloc
[37] Sagar Suri, artículo de discusión sobre las arquitecturas más influyentes de Flutter: https://medium.com/flutterpub/architecting-your-flutter-project-bd04e144a8f1
[38] Dart Streams, página de explicación: https://pub.dev/packages/streams_channel
[39] RxDart, librería de flutter: https://pub.dev/packages/rxdart
[40] States Rebuilder, artículo donde se comparan las distintas arquitecturas: https://medium.com/flutter-community/flutter-oneyearchallenge-scoped-model-vs-bloc-pattern-vs-states-rebuilder-23ba11813a4f
[41] Pub.dev, página oficial de librerías de terceros de Flutter: https://pub.dev/
[42] Librería Shared Preferencias de Flutter, página de la librería: https://pub.dev/packages/shared_preferences
[43] Librería Curved Navigation Bar de Flutter, página de la librería: https://pub.dev/packages/curved_navigation_bar
[44] Librería Path Provider de Flutter, página de la librería: https://pub.dev/packages/path_provider
[45] Librería Material Design Incons de Flutter, página de la librería: https://pub.dev/packages/material_design_icons_flutter
[46] Material Design, página oficial: https://material.io/design/
[47] Librería RFlutter Alert de Flutter, página de la librería: https://pub.dev/packages/rflutter_alert
[48] Librería MIME-Type de Flutter, página de la librería: https://pub.dev/packages/mime_type
[49] Librería Photo view de Flutter, página de la librería: https://pub.dev/packages/photo_view
[50] Librería Data Time Picker Formfield de Flutter, página de la librería: https://pub.dev/packages/datetime_picker_formfield
[51] Librería Folding Cell de Flutter, página de la librería: https://pub.dev/packages/folding_cell
[52] Librería Flutter Signature Pad de Flutter, página de la librería: https://pub.dev/packages/flutter_signature_pad
[53] Librería Flutter Advanced Network Image Provider de Flutter, página de la libería: https://pub.dev/packages/flutter_advanced_networkimage
[54] Librería Image Picker de Flutter, página de la librería: https://pub.dev/packages/image_picker
[55] Colores triádicos, artículo que lo define: https://cafeversatil.com/bricoydeco/colores-triadicos-en-decoracion/
[56] Postman, página oficial: https://www.getpostman.com/
[57] Bitrise, página oficial: https://app.bitrise.io/
[58] Integración continua, artículo de discusión: https://codeship.com/continuous-integration-essentials
[59] Entrega continua, artículo de Martin Fowler:

https://martinfowler.com/books/continuousDelivery.html

[60] Travis, página oficial: https://travis-ci.org/

[61] Jenkins, página oficial: https://jenkins.io/

[62] SQLite, página oficial: https://www.sqlite.org/index.html

[63] Swagger UI, página oficial: https://swagger.io/